facebook

Learning Embeddings: Word2vec and t-SNE

Deep Learning, February 23rd 2016, New York University

Laurens van der Maaten

Introduction

- You now know how to train a deep network on real-valued data vectors

Introduction

- You now know how to train a deep network on real-valued data vectors:
  - Do a forward pass of the data vector through the network
  - Compute the gradient of some loss function on the resulting output
  - Backpropagate this error through the network
  - Update the parameters using a (stochastic) gradient step

Introduction

- You now know how to train a deep network on real-valued data vectors:
  - Do a forward pass of the data vector through the network
  - Compute the gradient of some loss function on the resulting output
  - Backpropagate this error through the network
  - Update the parameters using a (stochastic) gradient step

- What if our training data is "discrete" in nature?
  - Language modeling: Predict a word, given its surrounding word
  - Representation learning: Find a representation for nodes of a graph

Language models

Language models

- Language models aim to predict a word given its surrounding words

Language models

- Language models aim to predict a word given its surrounding words
- In other words, they aim to build a distribution $p(\mathcal{W}) = p(w_1, w_2, \ldots, w_{|\mathcal{W}|})$

Language models

- Language models aim to predict a word given its surrounding words
- In other words, they aim to build a distribution $p(\mathcal{W}) = p(w_1, w_2, \ldots, w_{|\mathcal{W}|})$

- Standard language models are based on n-grams

Language models

- Language models aim to predict a word given its surrounding words
- In other words, they aim to build a distribution $p(\mathcal{W}) = p(w_1, w_2, \ldots, w_{|\mathcal{W}|})$

- Standard language models are based on n-grams:
  - The likelihood of a sentence: $p(\mathcal{W}) = \prod_{w_i \in \mathcal{W}} p(w_i | w_{i-1}, w_{i-2}, \ldots, w_{i-n})$
  - All of the probabilities are obtained by counting over a large corpus:

$$p(w_i | w_{i-1}, w_{i-2}) = \frac{C(w_i, w_{i-1}, w_{i-2})}{C(w_{i-1}, w_{i-2})}$$

Language models

- Example of using trigram model to compute the probability of a sentence:

$$p(\text{“}NYU\ is\ an\ excellent\ university\text{”}) =$$

Language models

- Example of using trigram model to compute the probability of a sentence:

$$p(\text{“}NYU\ is\ an\ excellent\ university\text{”}) = p(\text{“}NYU\text{”}) \times p(\text{“}is\text{”}|\text{“}NYU\text{”}) \times p(\text{“}an\text{”}|\text{“}NYU\text{”}, \text{“}is\text{”}) \\ \times p(\text{“}excellent\text{”}|\text{“}is\text{”}, \text{“}an\text{”}) \times p(\text{“}university\text{”}|\text{“}an\text{”}, \text{“}excellent\text{”})$$

Language models

- Example of using trigram model to compute the probability of a sentence:

$$p(\text{“}NYU\ is\ an\ excellent\ university\text{”}) = p(\text{“}NYU\text{”}) \times p(\text{“}is\text{”}|\text{“}NYU\text{”}) \times p(\text{“}an\text{”}|\text{“}NYU\text{”}, \text{“}is\text{”}) \times p(\text{“}excellent\text{”}|\text{“}is\text{”}, \text{“}an\text{”}) \times p(\text{“}university\text{”}|\text{“}an\text{”}, \text{“}excellent\text{”})$$

- To deal with non-observed trigrams, Kneser-Ney smoothing is often used
  - For bigrams, this smoother redefines the bigram probabilities as:

$$p_{KN}(w_t|w_{t-1}) = \frac{\max(C(w_{t-1}, w_t) - \delta, 0)}{\sum_{w'} C(w_{t-1}, w')} + \alpha p_{KN}(w_t)$$

  - This redistribution of (n-1)-gram to n-gram probabilities is applied recursively

Representing discrete objects

- An n-gram language model is basically a "handcrafted" model
- There is no learning beyond just counting how often patterns appear

Representing discrete objects

- An n-gram language model is basically a "handcrafted" model
- There is no learning beyond just counting how often patterns appear
- Learning representation like in deep networks may help here:
  - For instance, interchangeable words should have similar representations

Representing discrete objects

- An n-gram language model is basically a "handcrafted" model
- There is no learning beyond just counting how often patterns appear
- Learning representation like in deep networks may help here:
  - For instance, interchangeable words should have similar representations

- How can we represent elements of a discrete set in a network?

Representing discrete objects

- An n-gram language model is basically a "handcrafted" model
- There is no learning beyond just counting how often patterns appear
- Learning representation like in deep networks may help here:
  - For instance, interchangeable words should have similar representations

- How can we represent elements of a discrete set in a network?
  - Simple solution: use a "one-hot encoding"

One-hot encoding

- Represent each word as a vector of zeros, except for one element
- Set the value in the vector corresponding to the index in the set to 1:

```
vocabulary = (Monday, Tuesday, is, a, today)
Monday   = [          ]
Tuesday  = [          ]
is       = [          ]
a        = [          ]
today    = [          ]
```

* Example reproduced with permission from Mikolov.

One-hot encoding

- Represent each word as a vector of zeros, except for one element
- Set the value in the vector corresponding to the index in the set to 1:

```
vocabulary = (Monday, Tuesday, is, a, today)
Monday   = [1 0 0 0 0]
Tuesday  = [0 1 0 0 0]
is       = [0 0 1 0 0]
a        = [0 0 0 1 0]
today    = [0 0 0 0 1]
```

* Example reproduced with permission from Mikolov.

One-hot encoding

- Represent each word as a vector of zeros, except for one element
- Set the value in the vector corresponding to the index in the set to 1:

```
vocabulary = (Monday, Tuesday, is, a, today)
Monday   = [1 0 0 0 0]
Tuesday  = [0 1 0 0 0]
is       = [0 0 1 0 0]
a        = [0 0 0 1 0]
today    = [0 0 0 0 1]
```

- This is also known as a 1-of-K encoding (with K the vocabulary size)

* Example reproduced with permission from Mikolov.

Related: Bag-of-words representation

- A related representation is the bag-of-words representation for documents
- It simply sums one-hot representation over all words in the document:

```
vocabulary = (Monday, Tuesday, is, a, today)
Monday Monday       = [2 0 0 0 0]
today is a Monday   = [1 0 1 1 1]
today is a Tuesday  = [0 1 1 1 1]
is a Monday today   = [1 0 1 1 1]
```

* Example reproduced with permission from Mikolov.

Related: Bag-of-words representation

- A related representation is the bag-of-words representation for documents
- It simply sums one-hot representation over all words in the document:

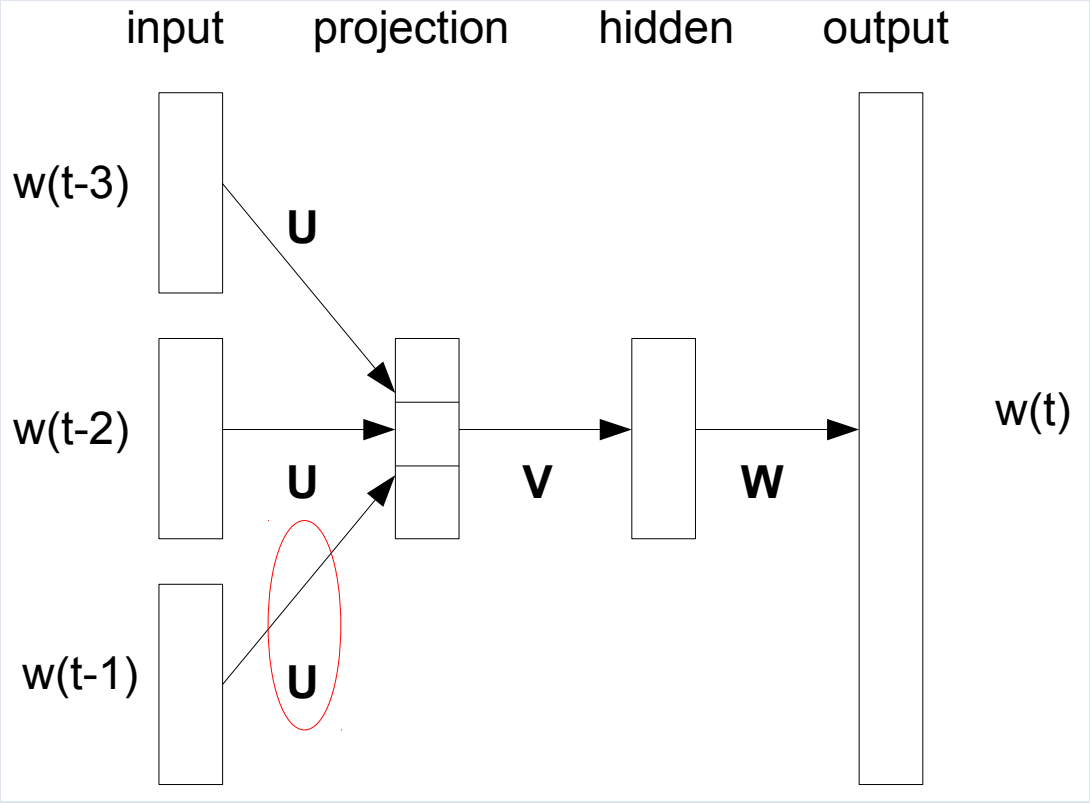
```
vocabulary = (Monday, Tuesday, is, a, today)
Monday Monday       = [2 0 0 0 0]
today is a Monday   = [1 0 1 1 1]
today is a Tuesday  = [0 1 1 1 1]
is a Monday today   = [1 0 1 1 1]
```

- Indeed, you could build bags-of-n-grams representations, too

* Example reproduced with permission from Mikolov.

A Feedforward Language Model

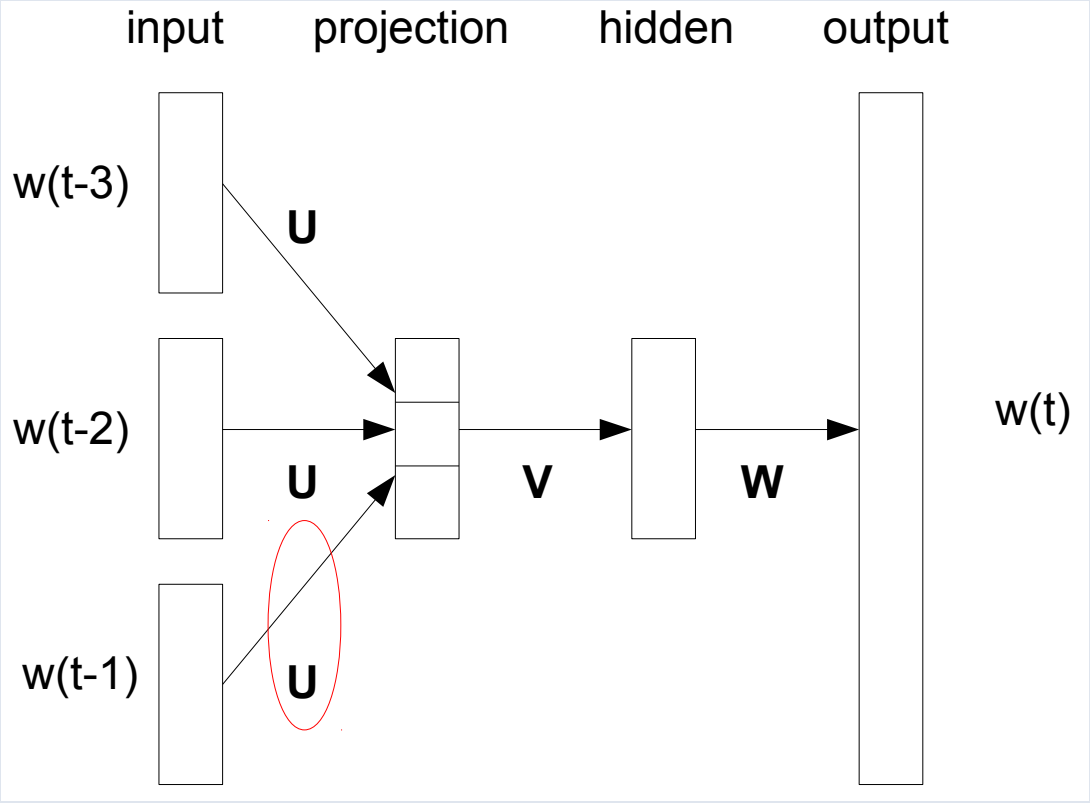
- We could use the following architecture for a word-prediction model:



* Figure reproduced with permission from Mikolov.

A Feedforward Language Model

- What does the matrix **U** actually model?



* Figure reproduced with permission from Mikolov.

Embedding

- Each column of **U** is an "embedding" of the corresponding word

Embedding

- Each column of **U** is an "embedding" of the corresponding word
- You can thus think of each word as being represented by a point that is "embedded" in a high-dimensional space

Embedding

- Each column of **U** is an "embedding" of the corresponding word
- You can thus think of each word as being represented by a point that is "embedded" in a high-dimensional space

- If the language model is trained well, then words that can be used interchangeably should have similar embeddings

Word2vec

- Word2vec is a very simple, very efficient language model:
  - It does not concatenate word embeddings, but it sums them
  - It does not use the second hidden layer
  - It does not use a multi-class logistic loss (softmax) over predictions

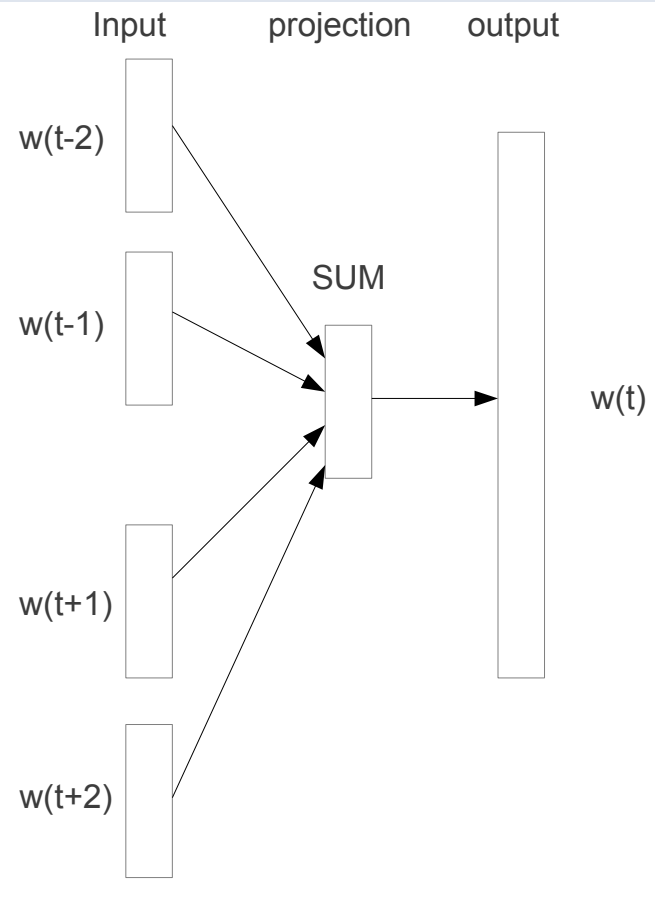
Word2vec

- Word2vec is a very simple, very efficient language model:
  - It does not concatenate word embeddings, but it sums them
  - It does not use the second hidden layer
  - It does not use a multi-class logistic loss (softmax) over predictions

- Because it is so simple, it can be trained on billions of words
- This has made it the de-facto standard in word embedding

Word2vec: Architectures

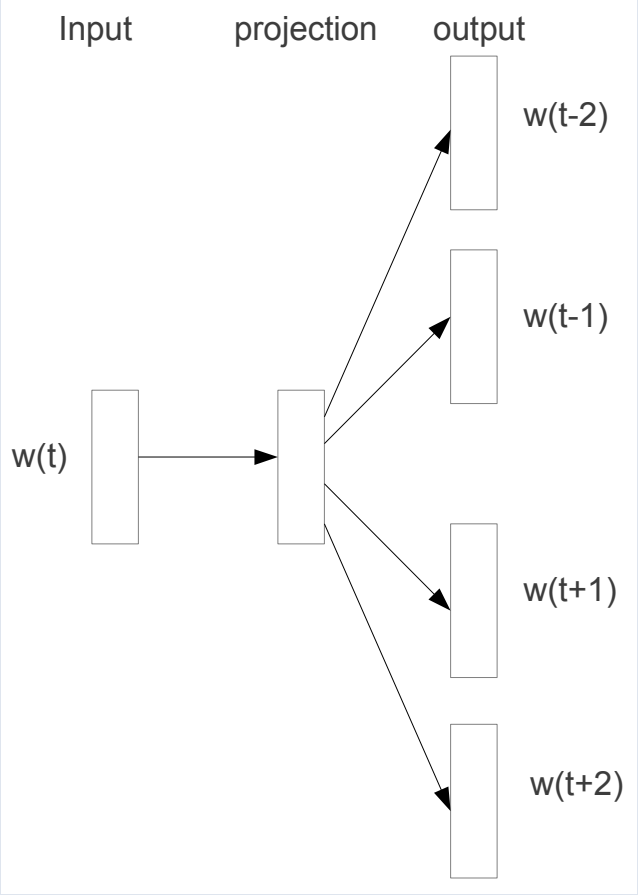
- "Continuous BoW" predicts current word given the surrounding words:



* Figure reproduced with permission from Mikolov.

Word2vec: Architectures

- "Skip-gram" predicts surrounding words given the current word:



* Figure reproduced with permission from Mikolov.

Loss function

- What loss are we optimizing in the word prediction?

Loss function

- What loss are we optimizing in the word prediction?

- A simple loss would be to minimize the multi-class logistic loss:

$$p(w_c|w_t) = \frac{\exp(\mathbf{u}_{w_t}^\top \mathbf{u}_{w_c})}{\sum_{w'} \exp(\mathbf{u}_{w_t}^\top \mathbf{u}_{w'})} \qquad \ell(\mathbf{U}) = -\sum_{c \in \mathcal{C}} \log p(w_c|w_t)$$

Loss function

- What loss are we optimizing in the word prediction?

- A simple loss would be to minimize the multi-class logistic loss:

$$p(w_c|w_t) = \frac{\exp(\mathbf{u}_{w_t}^\top \mathbf{u}_{w_c})}{\sum_{w'} \exp(\mathbf{u}_{w_t}^\top \mathbf{u}_{w'})} \qquad \ell(\mathbf{U}) = -\sum_{c \in \mathcal{C}} \log p(w_c|w_t)$$

softmax probability of predicting context word

Loss function

- What loss are we optimizing in the word prediction?

- A simple loss would be to minimize the multi-class logistic loss:

$$p(w_c|w_t) = \frac{\exp(\mathbf{u}_{w_t}^\top \mathbf{u}_{w_c})}{\sum_{w'} \exp(\mathbf{u}_{w_t}^\top \mathbf{u}_{w'})}$$

$$\ell(\mathbf{U}) = -\sum_{c \in \mathcal{C}} \log p(w_c|w_t)$$

(negative) log-probabilities summed over all words

Loss function

- What loss are we optimizing in the word prediction?

- A simple loss would be to minimize the multi-class logistic loss:

$$p(w_c|w_t) = \frac{\exp(\mathbf{u}_{w_t}^\top \mathbf{u}_{w_c})}{\sum_{w'} \exp(\mathbf{u}_{w_t}^\top \mathbf{u}_{w'})} \qquad \ell(\mathbf{U}) = -\sum_{c \in \mathcal{C}} \log p(w_c|w_t)$$

- Can you think of a problem when minimizing this loss?

Loss function

- What loss are we optimizing in the word prediction?

- A simple loss would be to minimize the multi-class logistic loss:

$$p(w_c|w_t) = \frac{\exp(\mathbf{u}_{w_t}^\top \mathbf{u}_{w_c})}{\sum_{w'} \exp(\mathbf{u}_{w_t}^\top \mathbf{u}_{w'})} \qquad \ell(\mathbf{U}) = -\sum_{c \in \mathcal{C}} \log p(w_c|w_t)$$

- Can you think of a problem when minimizing this loss?
  - The vocabulary may be very large! You will be waiting forever...

Word2vec: Loss function

- Word2vec minimizes a binary logistic loss on positive and negative samples:

$$\ell(\mathbf{U}) = \sum_{c \in \mathcal{C}} \log \sigma \left(\mathbf{u}_{w_t}^\top \mathbf{u}_{w_c} \right) + \sum_{j=1}^{K} \log \sigma \left(-\mathbf{u}_{w_t}^\top \mathbf{u}_{v'} \right) \text{ with } v' \sim P(\mathcal{V})$$

Word2vec: Loss function

- Word2vec minimizes a binary logistic loss on positive and negative samples:

$$\ell(\mathbf{U}) = \sum_{c \in \mathcal{C}} \log \sigma\left(\mathbf{u}_{w_t}^\top \mathbf{u}_{w_c}\right) + \sum_{j=1}^{K} \log \sigma\left(-\mathbf{u}_{w_t}^\top \mathbf{u}_{v'}\right) \text{ with } v' \sim P(\mathcal{V})$$

"positive" part

"negative" part

Word2vec: Loss function

- Word2vec minimizes a binary logistic loss on positive and negative samples:

$$\ell(\mathbf{U}) = \sum_{c \in \mathcal{C}} \log \sigma\left(\mathbf{u}_{w_t}^\top \mathbf{u}_{w_c}\right) + \sum_{j=1}^{K} \log \sigma\left(-\mathbf{u}_{w_t}^\top \mathbf{u}_{v'}\right) \quad \text{with } v' \sim P(\mathcal{V})$$

encourage current word and context word to have a large inner product

Word2vec: Loss function

- Word2vec minimizes a binary logistic loss on positive and negative samples:

$$\ell(\mathbf{U}) = \sum_{c \in \mathcal{C}} \log \sigma\left(\mathbf{u}_{w_t}^\top \mathbf{u}_{w_c}\right) + \sum_{j=1}^{K} \log \sigma\left(-\mathbf{u}_{w_t}^\top \mathbf{u}_{v'}\right) \text{ with } v' \sim P(\mathcal{V})$$

for all words in the context

Word2vec: Loss function

- Word2vec minimizes a binary logistic loss on positive and negative samples:

$$\ell(\mathbf{U}) = \sum_{c \in \mathcal{C}} \log \sigma\left(\mathbf{u}_{w_t}^\top \mathbf{u}_{w_c}\right) + \sum_{j=1}^{K} \log \sigma\left(-\mathbf{u}_{w_t}^\top \mathbf{u}_{v'}\right) \quad \text{with} \; \boxed{v' \sim P(\mathcal{V})}$$

pick a random word according to the unigram distribution

Word2vec: Loss function

- Word2vec minimizes a binary logistic loss on positive and negative samples:

$$\ell(\mathbf{U}) = \sum_{c \in \mathcal{C}} \log \sigma\left(\mathbf{u}_{w_t}^\top \mathbf{u}_{w_c}\right) + \sum_{j=1}^{K} \log \sigma\left(-\mathbf{u}_{w_t}^\top \mathbf{u}_{v'}\right) \text{ with } v' \sim P(\mathcal{V})$$

encourage this random word to have a small inner product with the current word

Word2vec: Loss function

- Word2vec minimizes a binary logistic loss on positive and negative samples:

$$\ell(\mathbf{U}) = \sum_{c \in \mathcal{C}} \log \sigma\left(\mathbf{u}_{w_t}^\top \mathbf{u}_{w_c}\right) + \sum_{j=1}^{K} \log \sigma\left(-\mathbf{u}_{w_t}^\top \mathbf{u}_{v'}\right) \text{ with } v' \sim P(\mathcal{V})$$

for K randomly picked words

Word2vec: Loss function

- Word2vec minimizes a binary logistic loss on positive and negative samples:

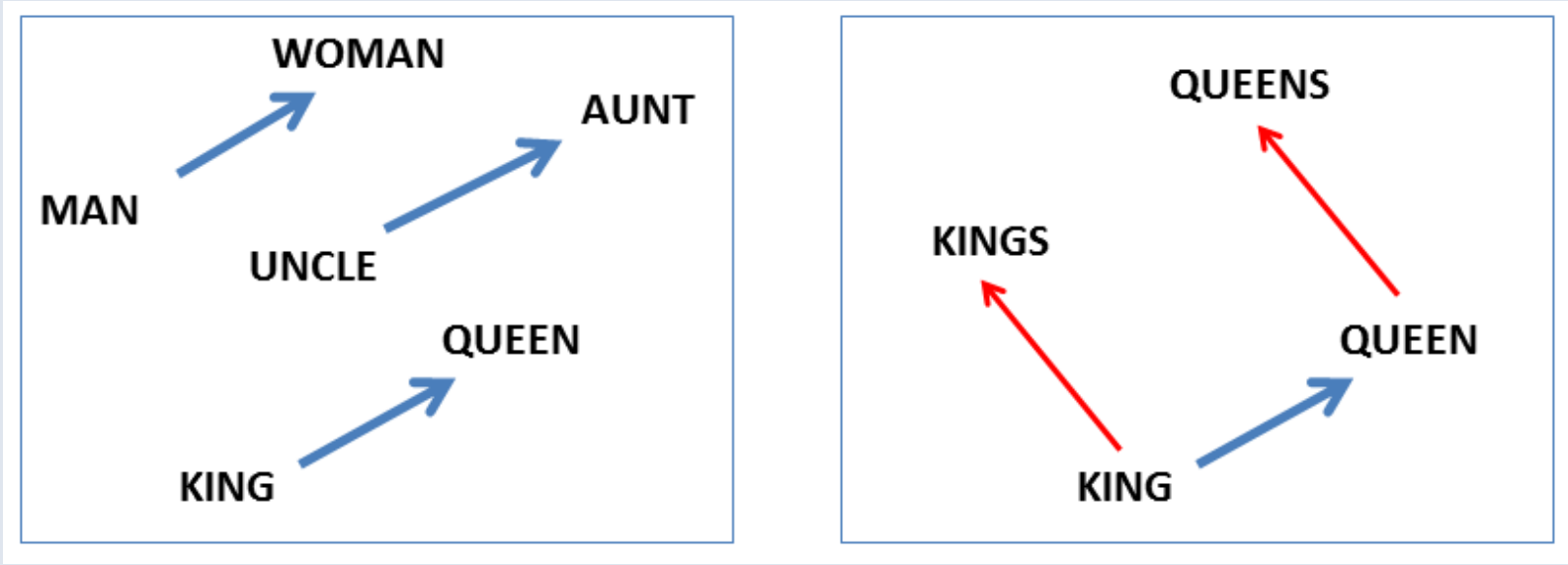
$$\ell(\mathbf{U}) = \sum_{c \in \mathcal{C}} \log \sigma \left(\mathbf{u}_{w_t}^\top \mathbf{u}_{w_c} \right) + \sum_{j=1}^{K} \log \sigma \left(-\mathbf{u}_{w_t}^\top \mathbf{u}_{v'} \right) \text{ with } v' \sim P(\mathcal{V})$$

- Learning with SGD in this loss is very efficient:
  - Take a word and its context from a text corpus
  - Sample K words (typically, 5<K<20) from the unigram distribution
  - Compute the loss and the (sparse!) gradient update

- Multithreaded implementation allows for training speed of 5M words / sec

Linguistic regularities

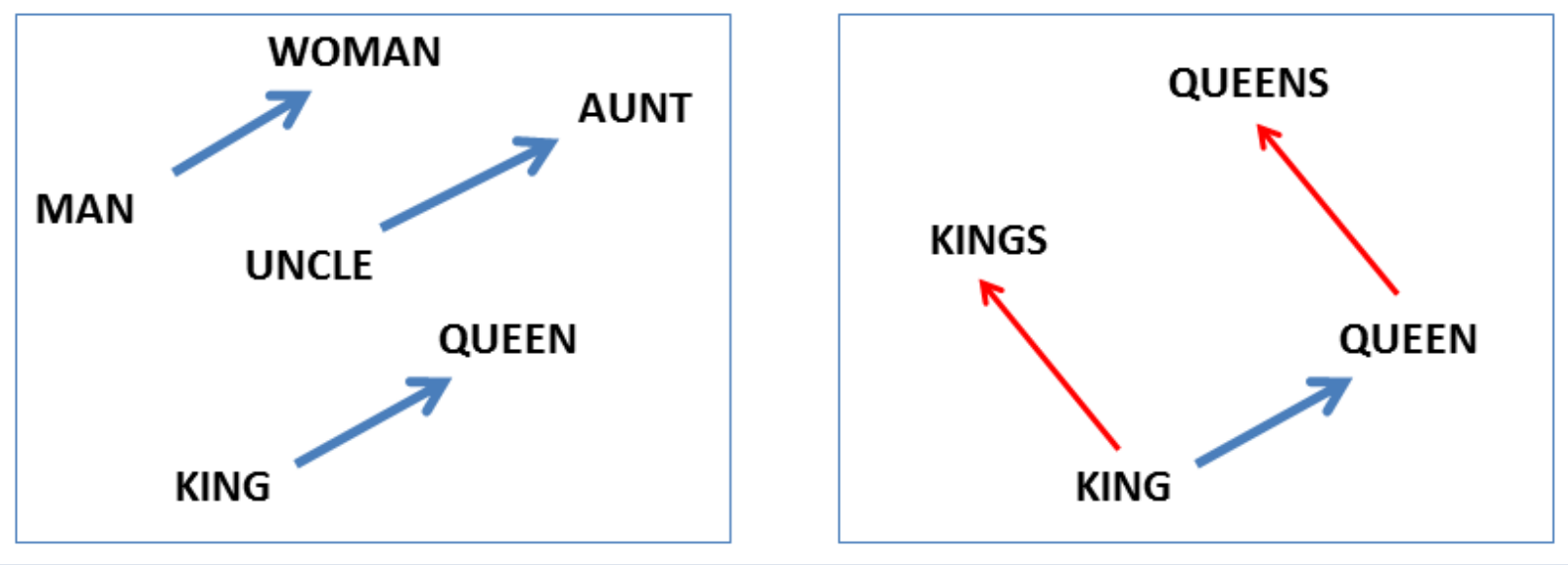
- Vector space implicitly encode regularities among words:



* Figure reproduced with permission from Mikolov.

Linguistic regularities

- Vector space implicitly encode regularities among words:



- We can exploit these regularities to do "linguistic arithmetic"

* Figure reproduced with permission from Mikolov.

Linguistic regularities

- Make dataset with analogies: "A is to B like C is to D"
- Answer the question "A is to B like C is to ?" by finding the word embedding that is closest to the embedding of B - A + C

Model	*Vector Dimensionality*	*Training Words*	*Training Time*	*Accuracy [%]*
Collobert NNLM	50	660M	2 months	11
Turian NNLM	200	37M	few weeks	2
Mnih NNLM	100	37M	7 days	9
Mikolov RNNLM	640	320M	weeks	25
Huang NNLM	50	990M	weeks	13
Our NNLM	100	6B	2.5 days	51
Skip-gram (hier.s.)	1000	6B	hours	66
CBOW (negative)	300	1.5B	**minutes**	**72**

* Table reproduced with permission from Mikolov.

Linguistic regularities

- Some examples of regularities:

Expression	*Nearest token*
Paris - France + Italy	Rome
bigger - big + cold	colder
sushi - Japan + Germany	bratwurst
Cu - copper + gold	Au
Windows - Microsoft + Google	Android
Montreal Canadiens - Montreal + Toronto	Toronto Maple Leafs

* Table reproduced with permission from Mikolov.

Linguistic regularities

- Compositionally by vector addition:

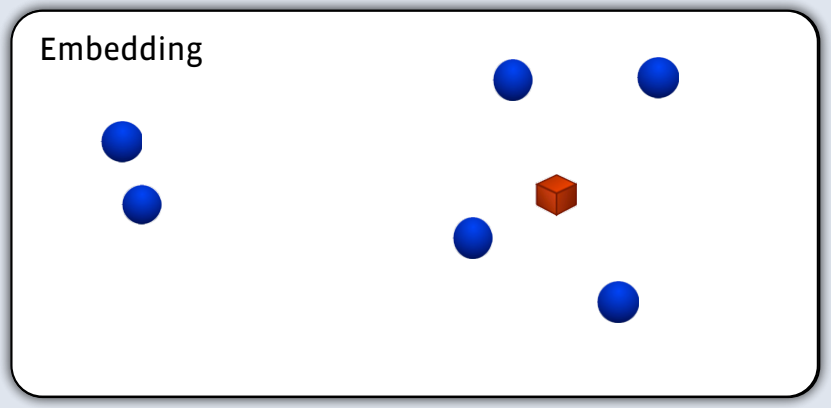
Expression	*Nearest tokens*
Czech + currency	koruna, Czech crown, Polish zloty, CTK
Vietnam + capital	Hanoi, Ho Chi Minh City, Viet Nam, Vietnamese
German + airlines	airline Lufthansa, carrier Lufthansa, flag carrier Lufthansa
Russian + river	Moscow, Volga River, upriver, Russia
French + actress	Juliette Binoche, Vanessa Paradis, Charlotte Gainsbourg

* Table reproduced with permission from Mikolov.

Visualizing graphs

Introduction

- Presume we are given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
- How do we learn a representation (embedding) for the vertices $v \in \mathcal{V}$?
  - We would like vertices that are connected to have similar embeddings
  - These embeddings we could then use in learning deep networks
  - If the embeddings are low-dimensional: use them to visualize the graph!



t-Stochastic Neighbor Embedding

- Convert the graph to a probability distribution over vertices:

$$p_{ij} = \frac{\exp(e_{ij})}{\sum_{k \neq l} \exp(e_{kl})}$$

- Strongly connected nodes will have large probabilities p_{ij}

t-Stochastic Neighbor Embedding

- Convert the graph to a probability distribution over vertices:

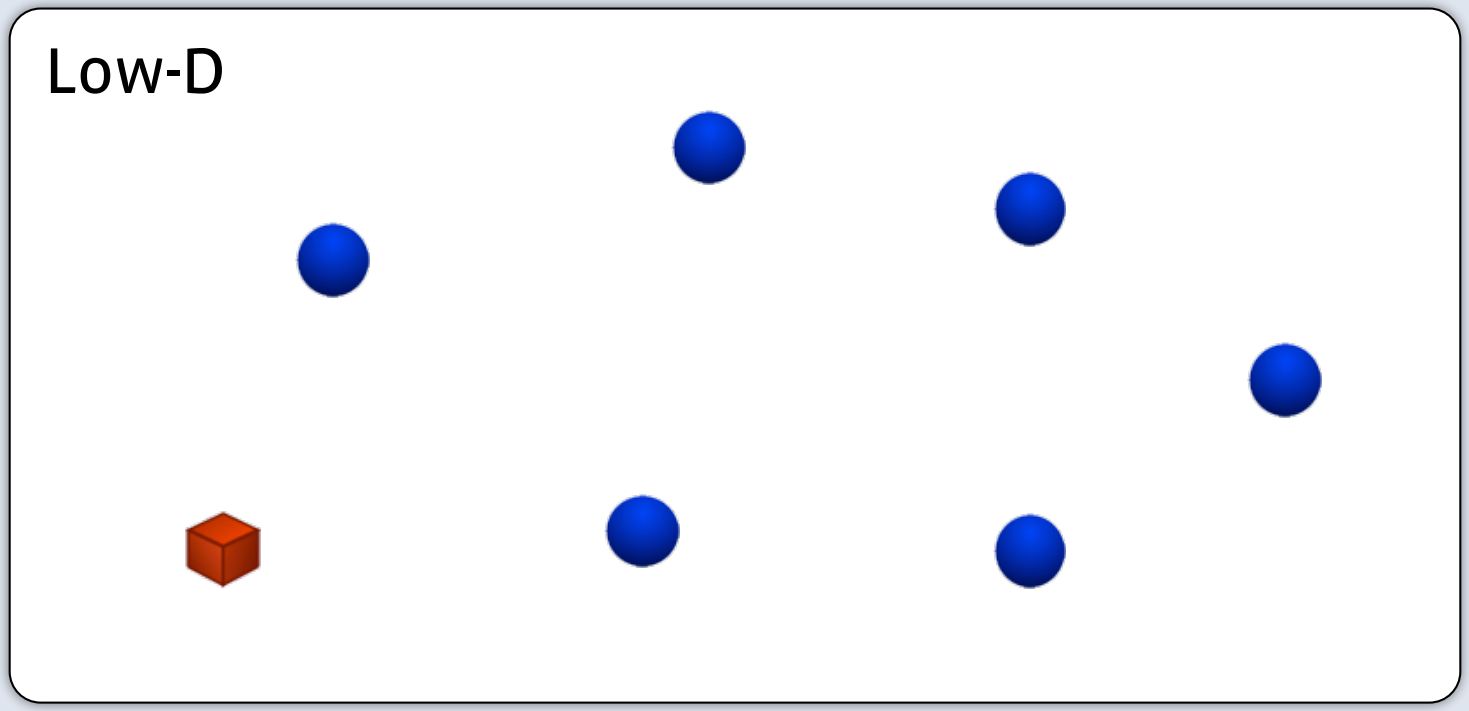
$$p_{ij} = \frac{\exp(e_{ij})}{\sum_{k \neq l} \exp(e_{kl})}$$

- Strongly connected nodes will have large probabilities p_{ij}

- When given high-dimensional data, we can compute similar probabilities:
  - For instance, set $e_{ij} = -\|\mathbf{x}_i - \mathbf{x}_j\|^2$ with $\mathbf{x}_i \in \mathbb{R}^D$
  - Note that this is essentially a (normalized) Gaussian kernel matrix

t-Stochastic Neighbor Embedding

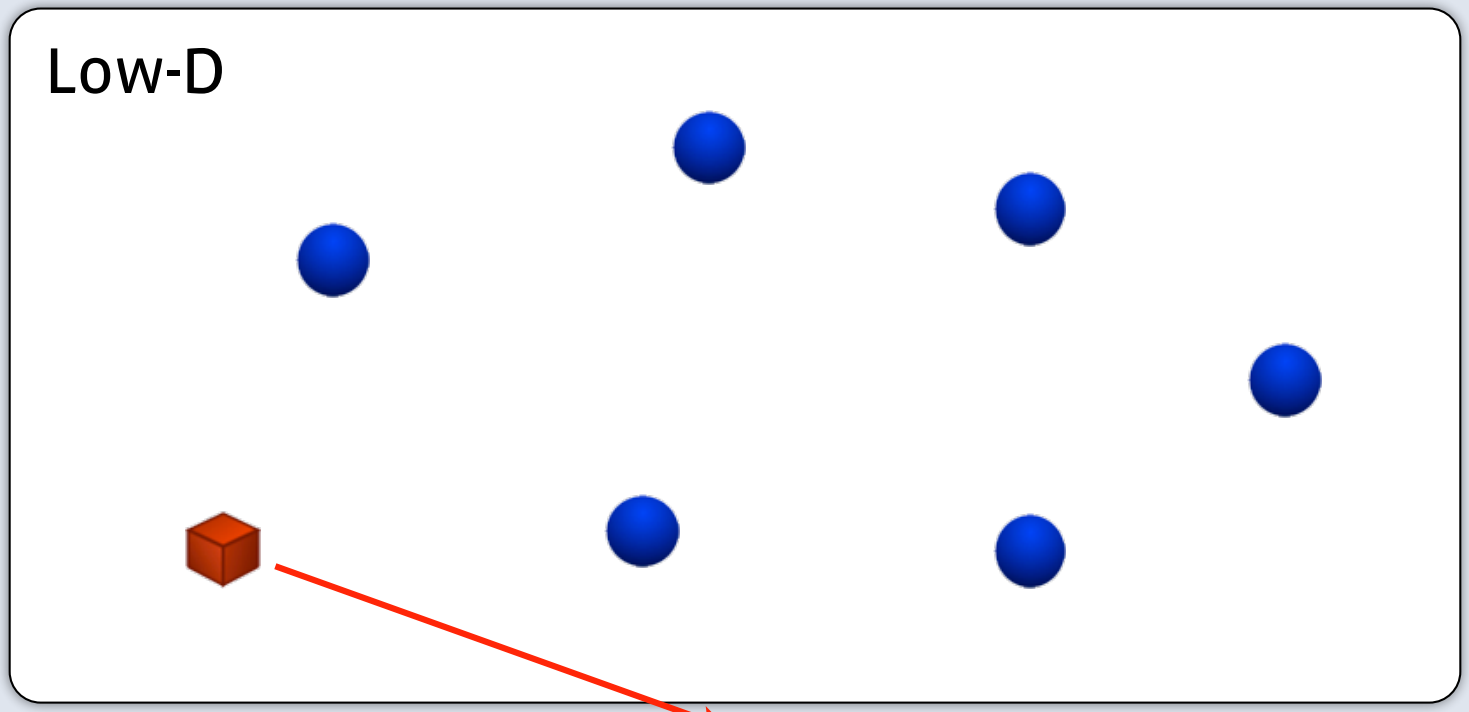
- Measure pairwise similarities between the embeddings:



$$q_{ij} = \frac{(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1}}{\sum_k \sum_{l \neq k} (1 + \|\mathbf{y}_k - \mathbf{y}_l\|^2)^{-1}}$$

t-Stochastic Neighbor Embedding

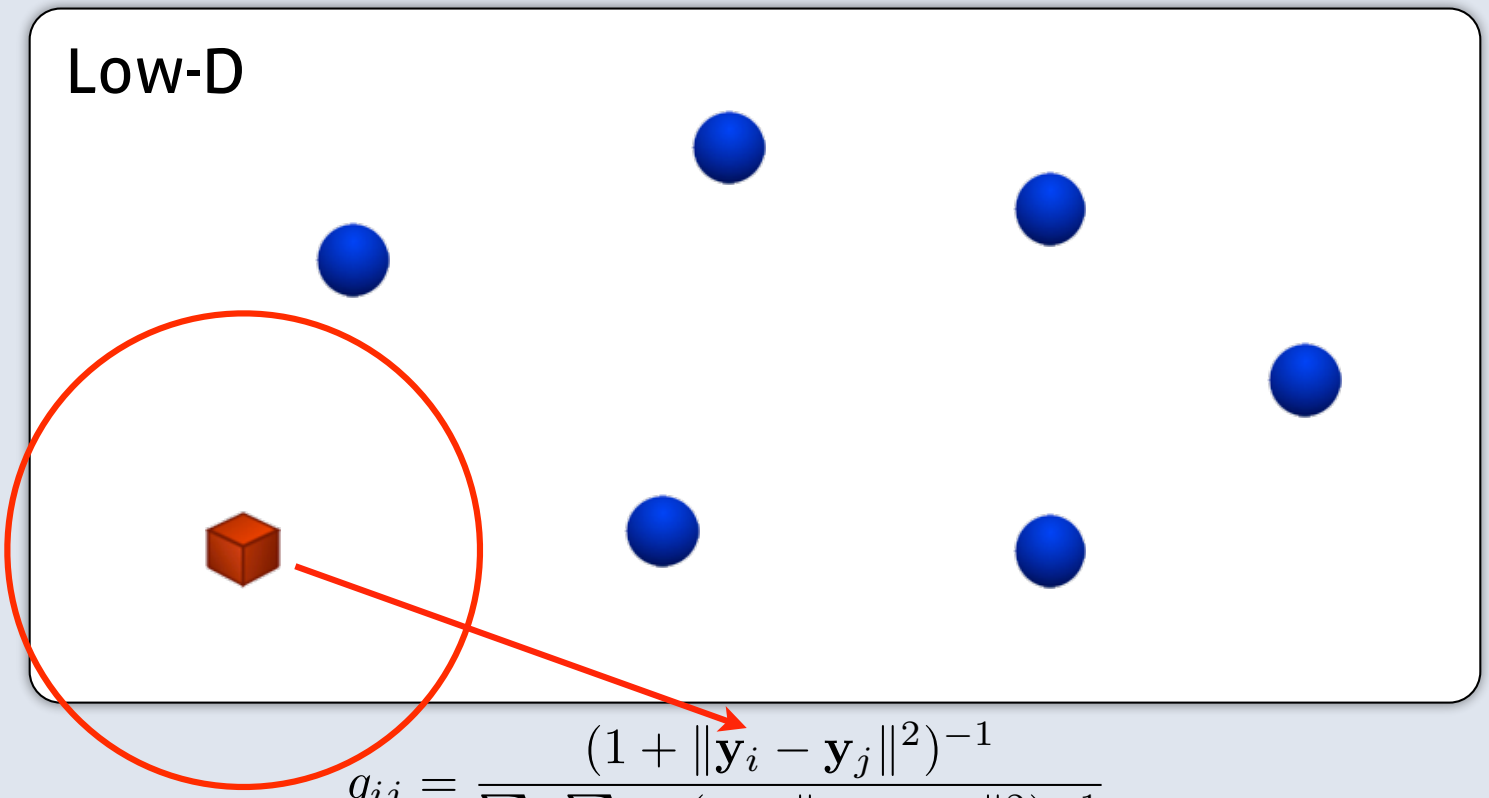
- Measure pairwise similarities between the embeddings:



$$q_{ij} = \frac{(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1}}{\sum_k \sum_{l \neq k} (1 + \|\mathbf{y}_k - \mathbf{y}_l\|^2)^{-1}}$$

t-Stochastic Neighbor Embedding

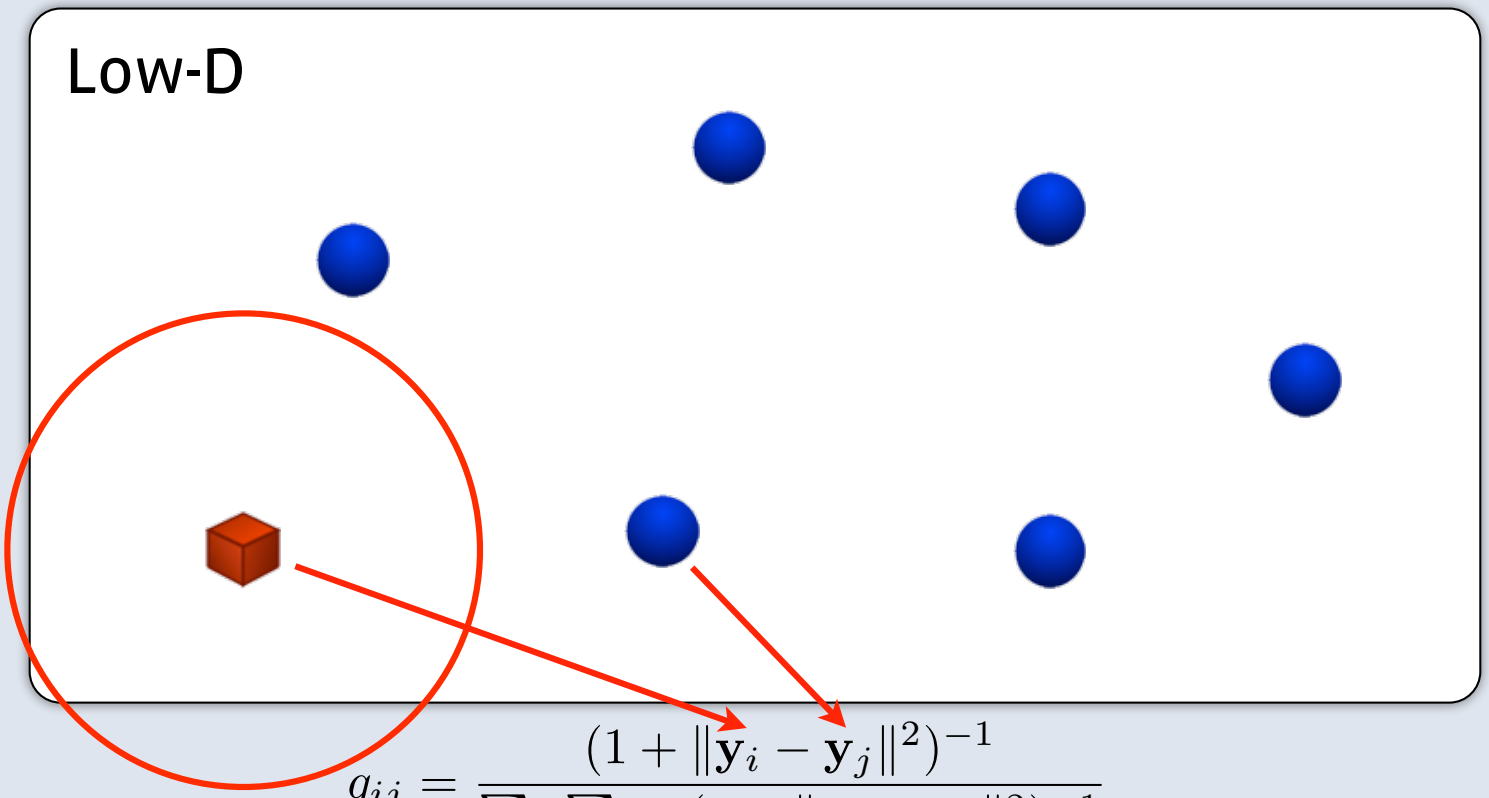
- Measure pairwise similarities between the embeddings:



$$q_{ij} = \frac{(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1}}{\sum_k \sum_{l \neq k} (1 + \|\mathbf{y}_k - \mathbf{y}_l\|^2)^{-1}}$$

t-Stochastic Neighbor Embedding

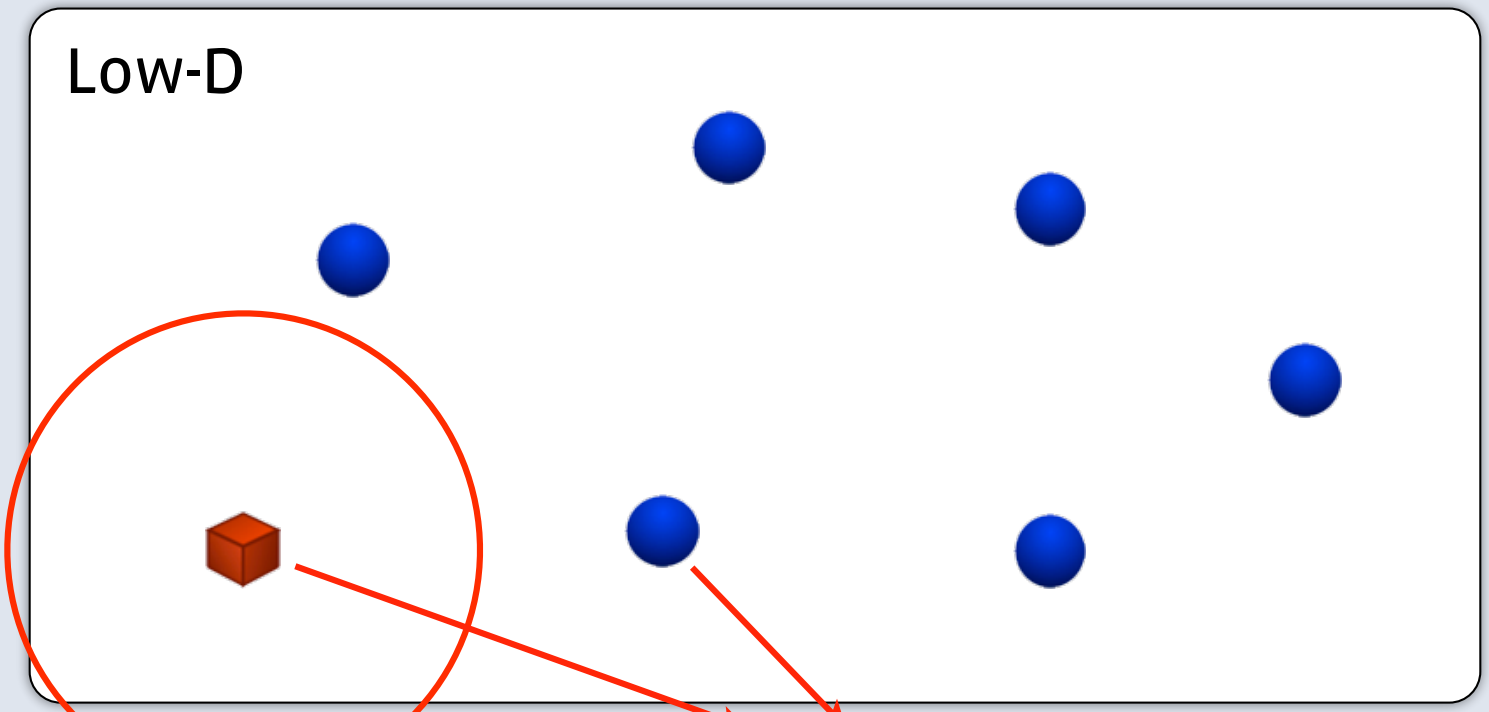
- Measure pairwise similarities between the embeddings:



$$q_{ij} = \frac{(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1}}{\sum_k \sum_{l \neq k} (1 + \|\mathbf{y}_k - \mathbf{y}_l\|^2)^{-1}}$$

t-Stochastic Neighbor Embedding

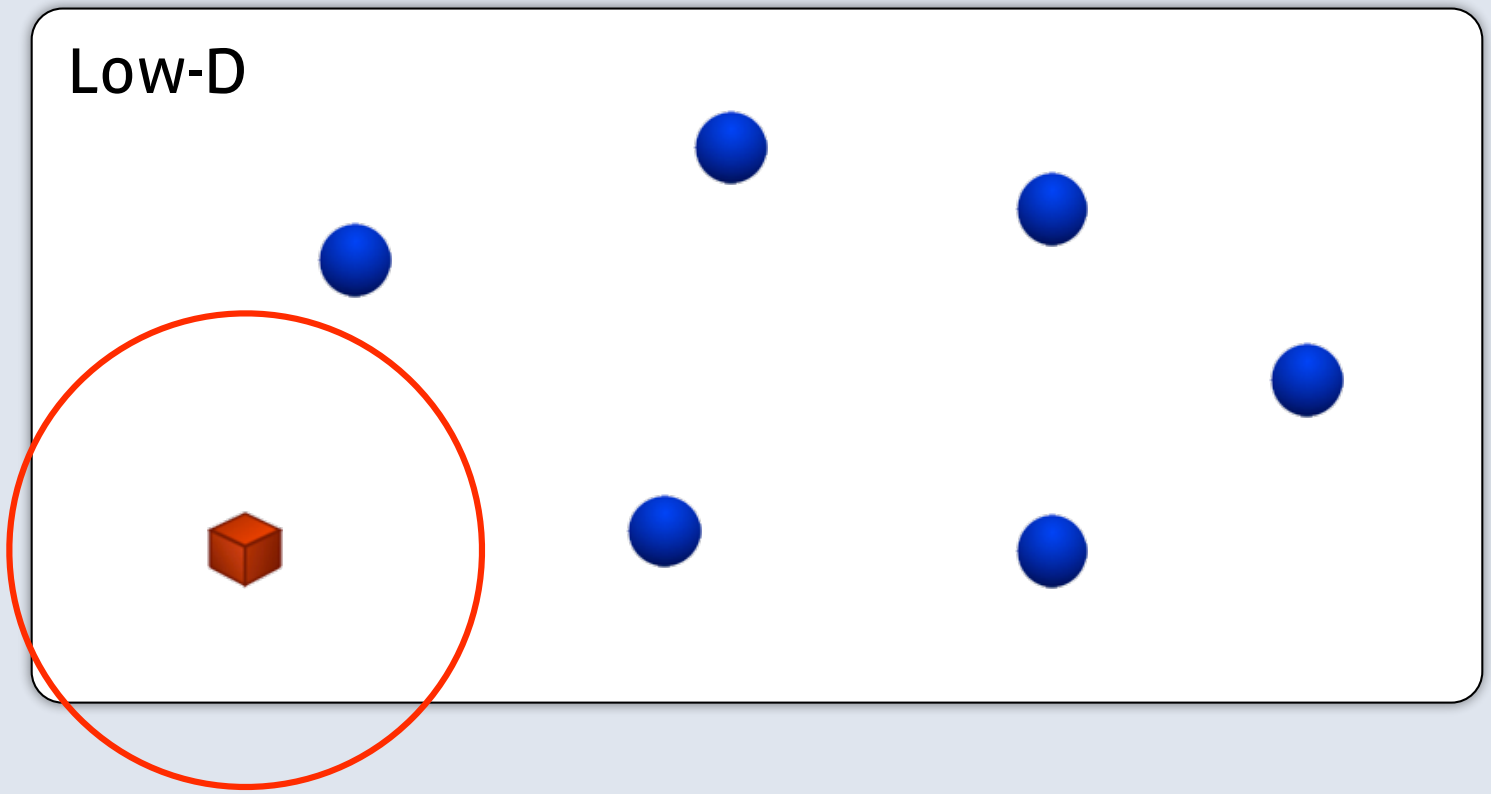
- Measure pairwise similarities between the embeddings:



$$q_{ij} = \frac{(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1}}{\sum_k \sum_{l \neq k} (1 + \|\mathbf{y}_k - \mathbf{y}_l\|^2)^{-1}}$$

t-Stochastic Neighbor Embedding

- Move points around to minimize: $KL(P||Q) = \sum_i \sum_{j \neq i} p_{ij} \log \frac{p_{ij}}{q_{ij}}$



t-Stochastic Neighbor Embedding

- Kullback-Leibler divergence: $KL(P||Q) = \sum_i \sum_{j \neq i} p_{ij} \log \frac{p_{ij}}{q_{ij}}$

t-Stochastic Neighbor Embedding

- Kullback-Leibler divergence: $KL(P||Q) = \sum_i \sum_{j \neq i} p_{ij} \log \frac{p_{ij}}{q_{ij}}$
  - Large p_{ij} modeled by small q_{ij}? Big penalty!
  - Small p_{ij} modeled by large q_{ij}? Not-so-big penalty.

t-Stochastic Neighbor Embedding

- Kullback-Leibler divergence: $KL(P||Q) = \sum_i \sum_{j \neq i} p_{ij} \log \frac{p_{ij}}{q_{ij}}$
  - Large p_{ij} modeled by small q_{ij}? Big penalty!
  - Small p_{ij} modeled by large q_{ij}? Not-so-big penalty.
- t-SNE makes sure connected vertices are close together in the embedding!

Visualization experiment

- Suppose we are given the MNIST dataset of handwritten digits

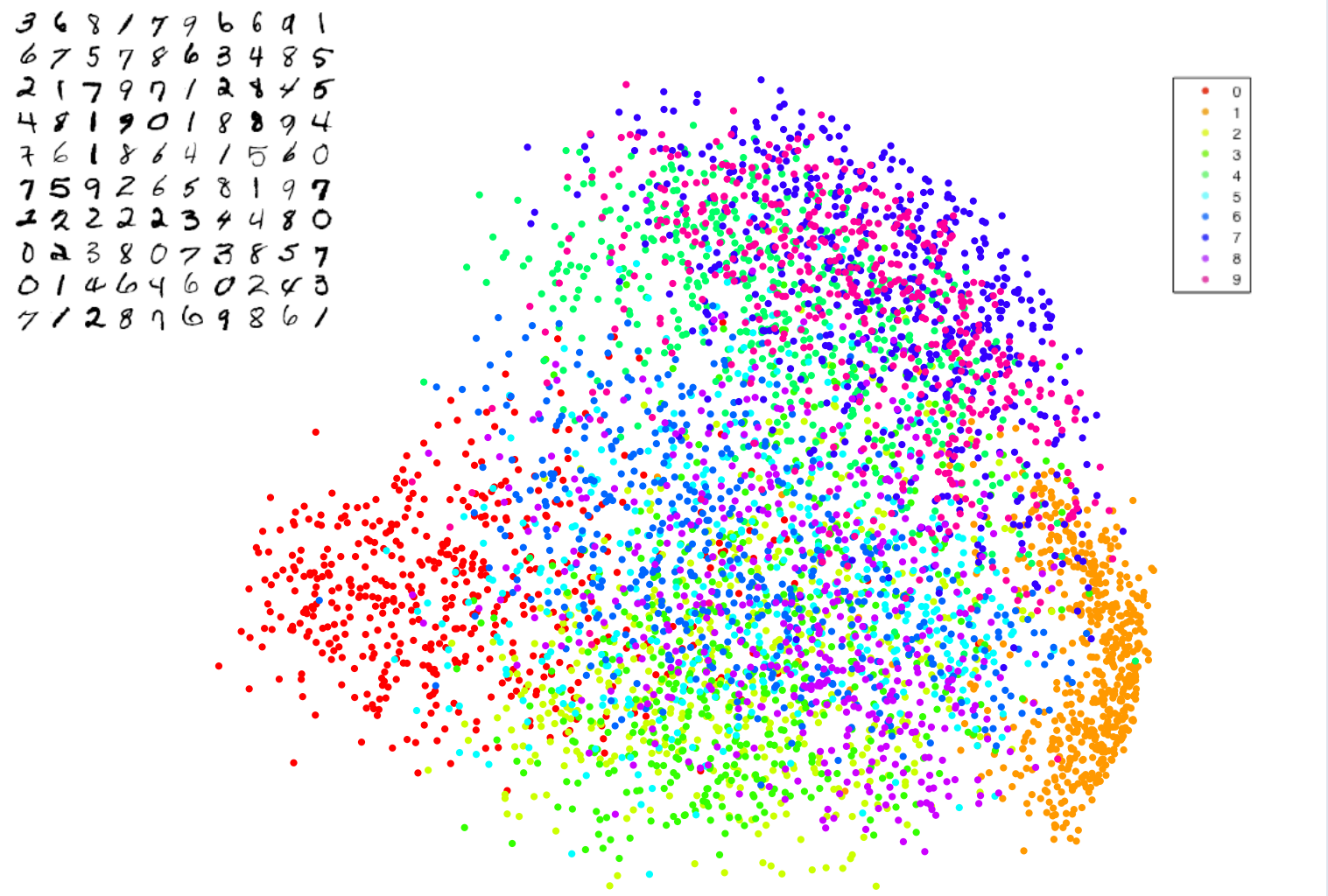
- Can we make a scatter plot that shows some of the structure in this data?

Visualization experiment

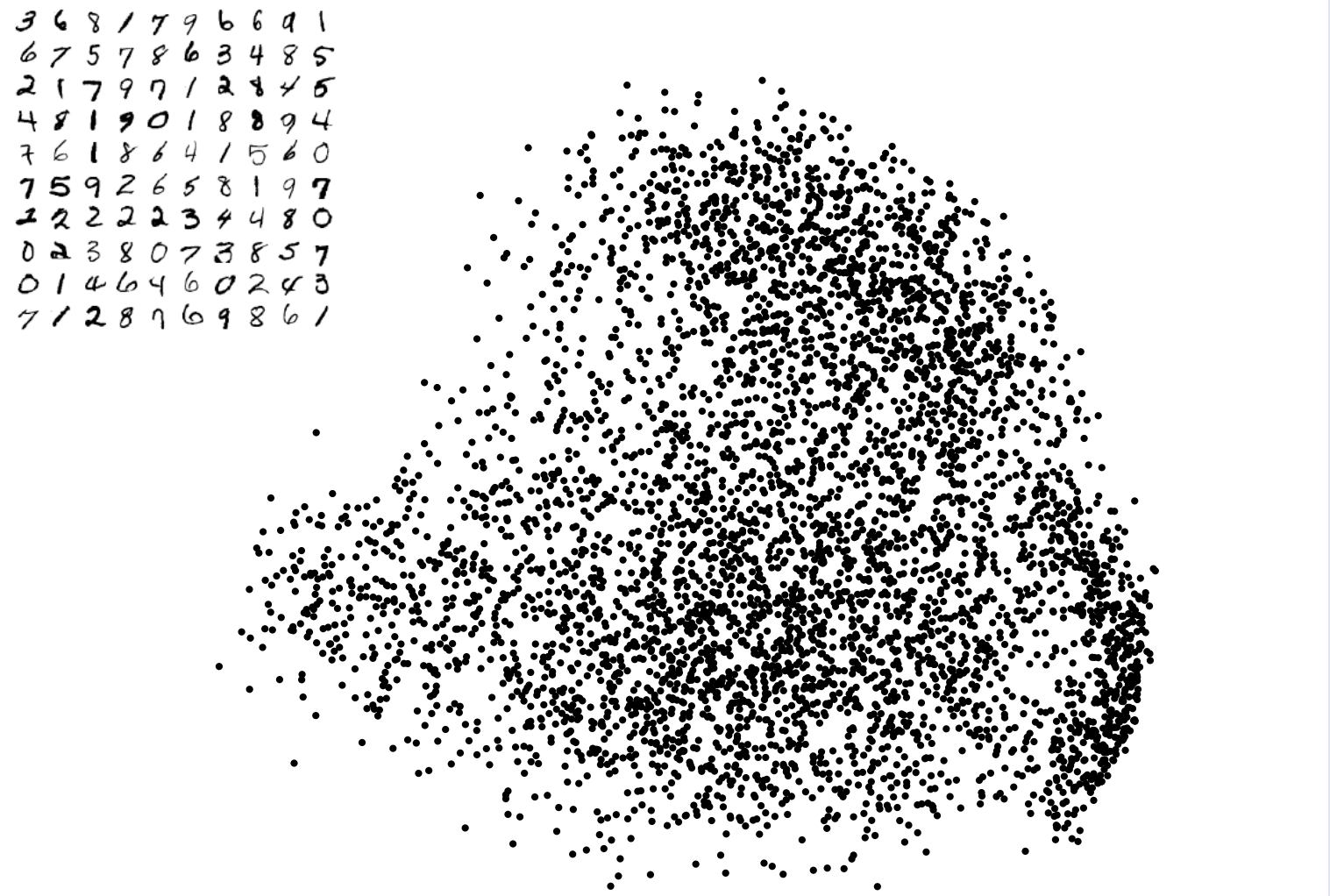
- Suppose we are given the MNIST dataset of handwritten digits

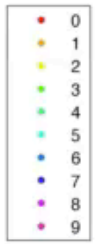
- Can we make a scatter plot that shows some of the structure in this data?
  - Approach 1:
    - Apply PCA on the digits and make a scatter plot in which the data is projected onto its first two principal components
  - Approach 2:
    - Construct a k-nearest neighbor graph (or simply compute a Gaussian kernel) and run t-SNE to learn a 2D embedding that you can show in a scatter plot

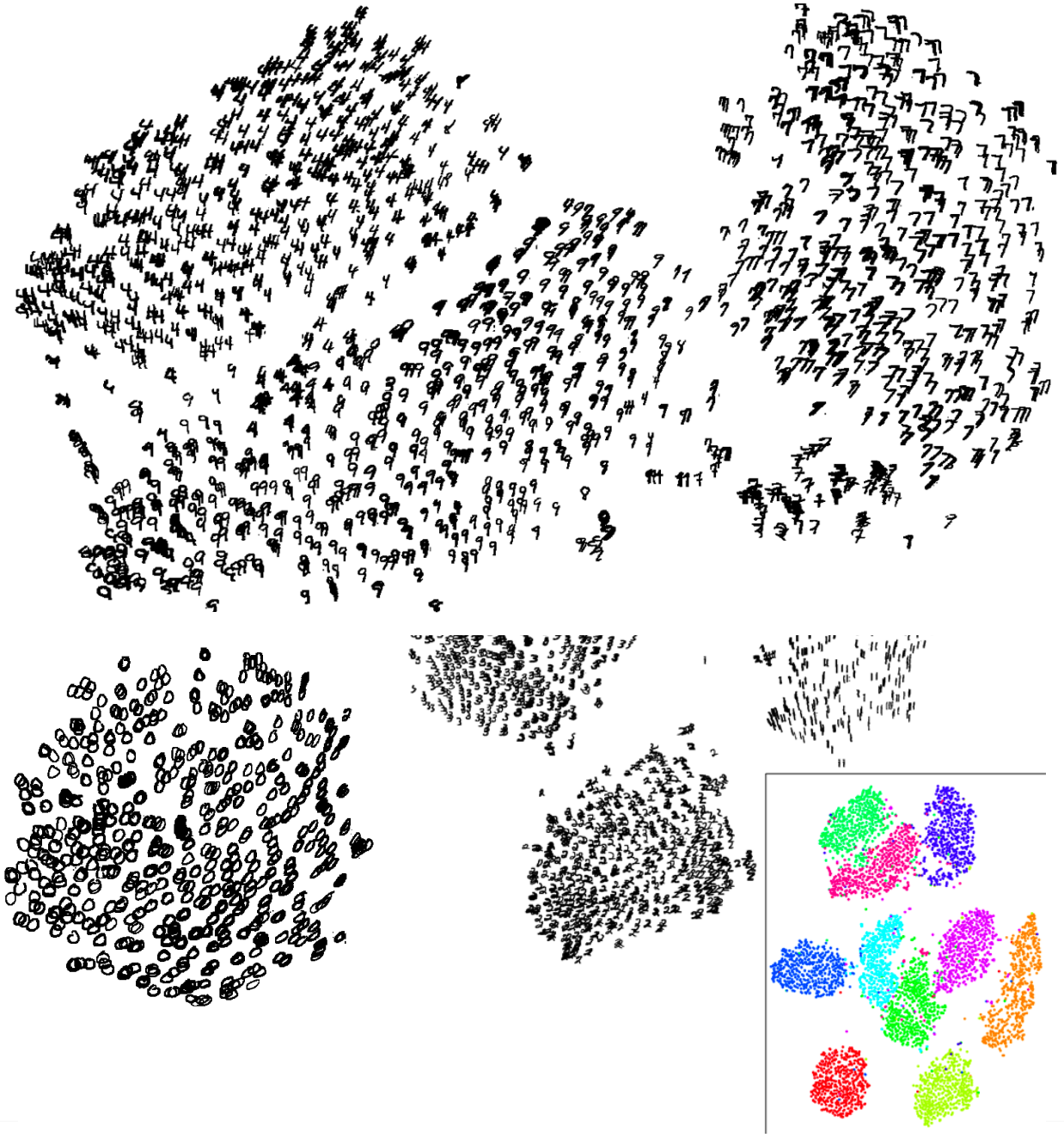
Principal Components Analysis

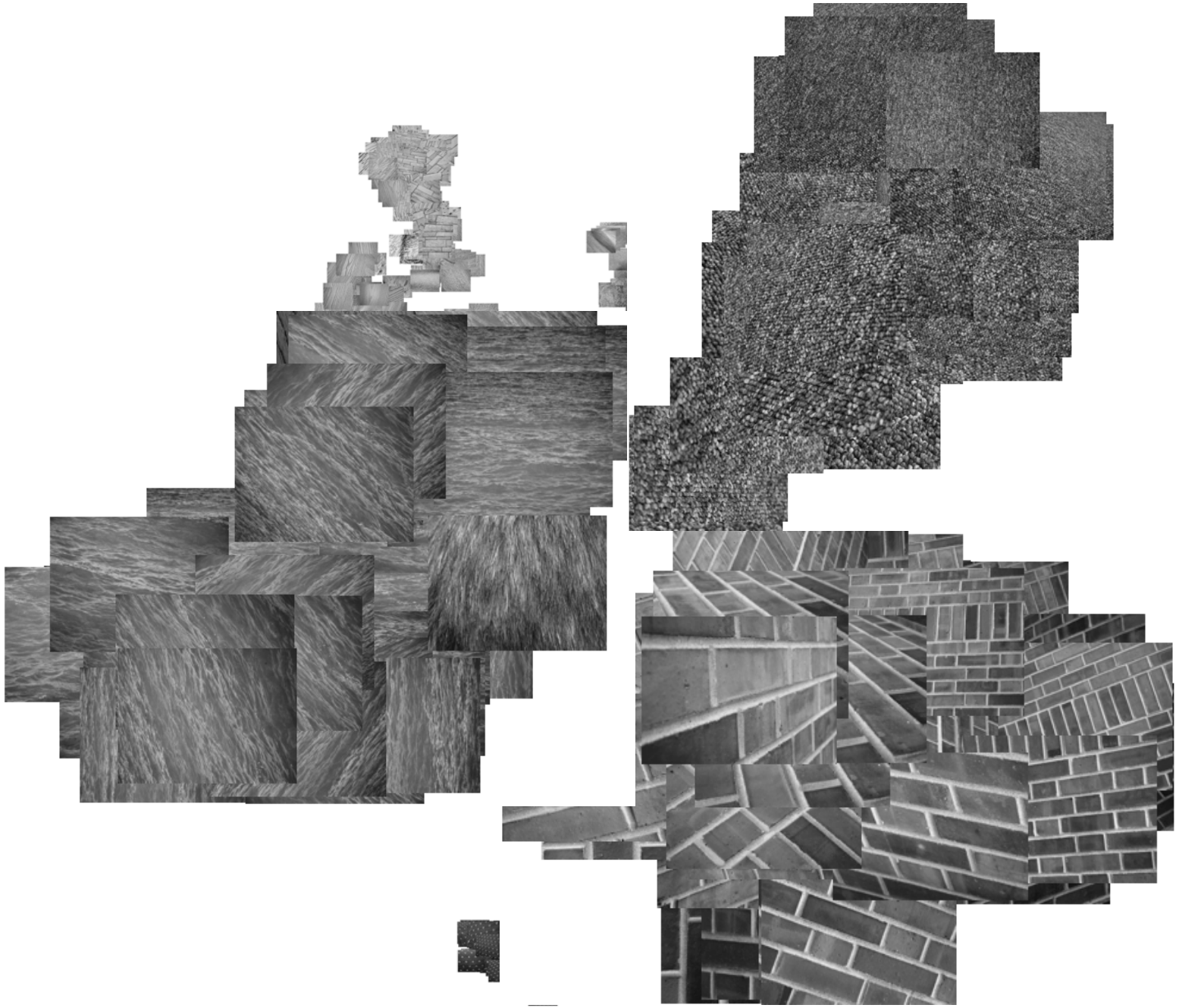


Principal Components Analysis



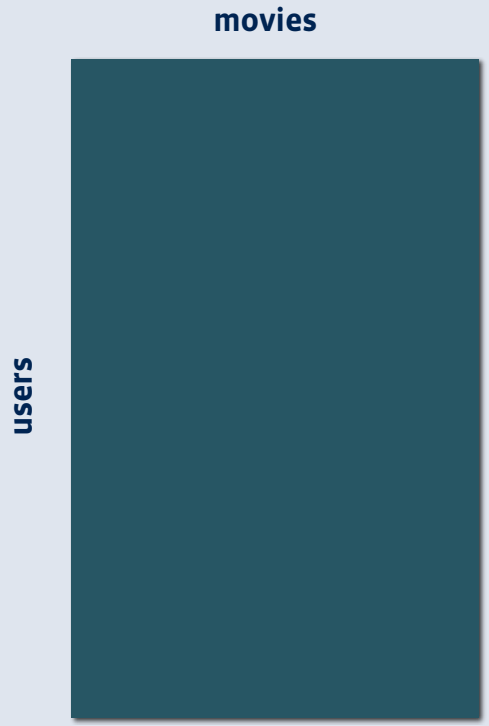

0
1
2
3
4
5
6
7
8
9





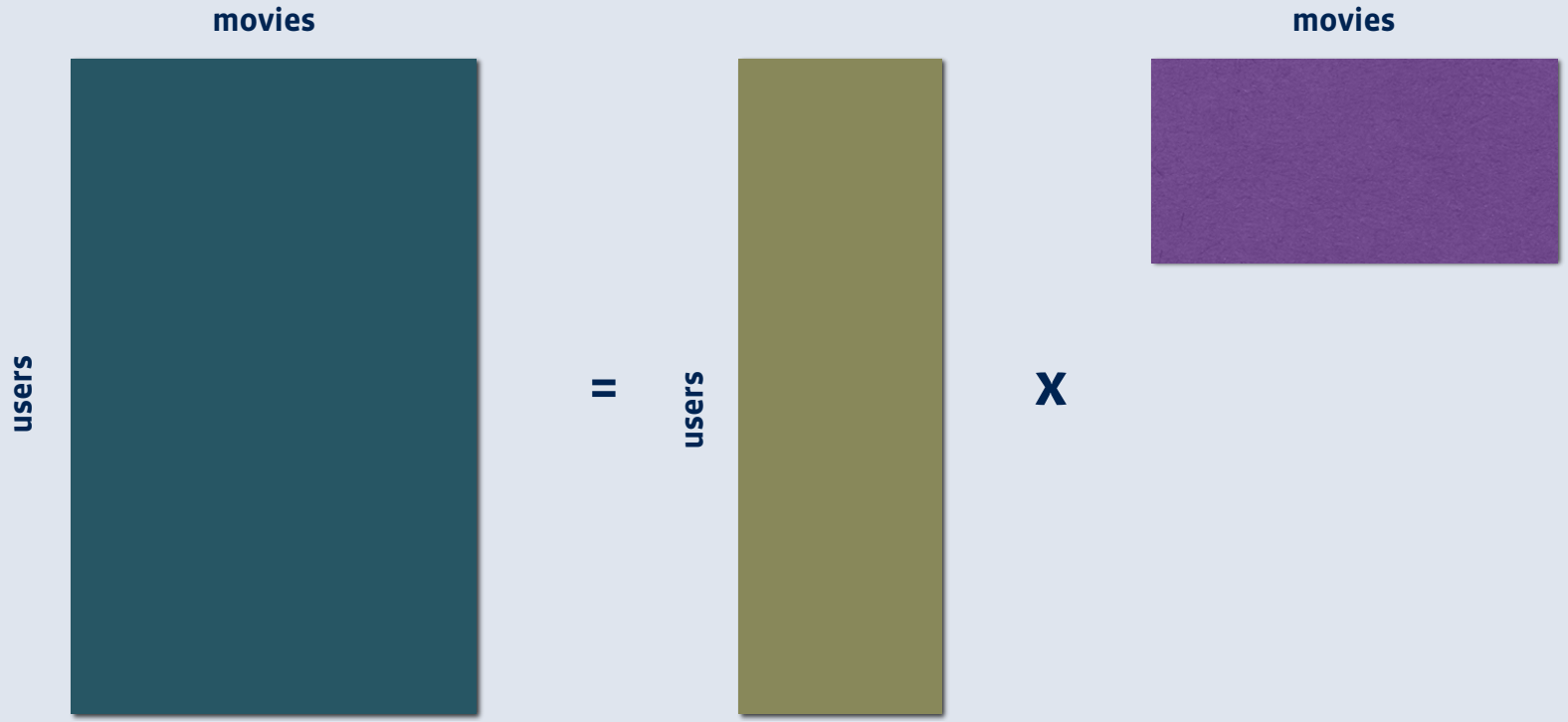
Visualizing movies

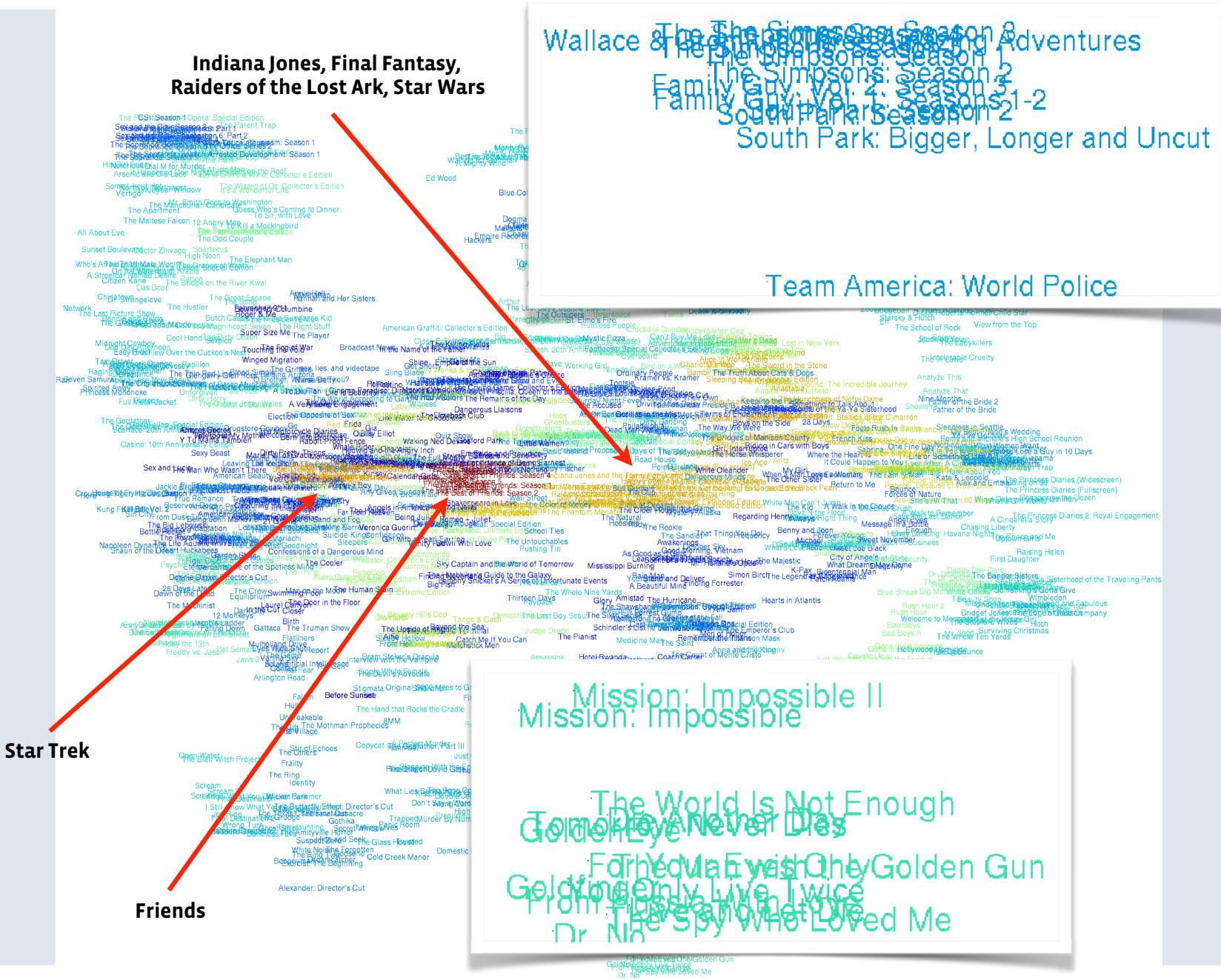
- Suppose you have a collection of user-movie ratings in a large *rating matrix*



Visualizing movies

- Suppose you have a collection of user-movie ratings in a large *rating matrix*
- *Decompose* the rating matrix to obtain *user features* and *movie features*:

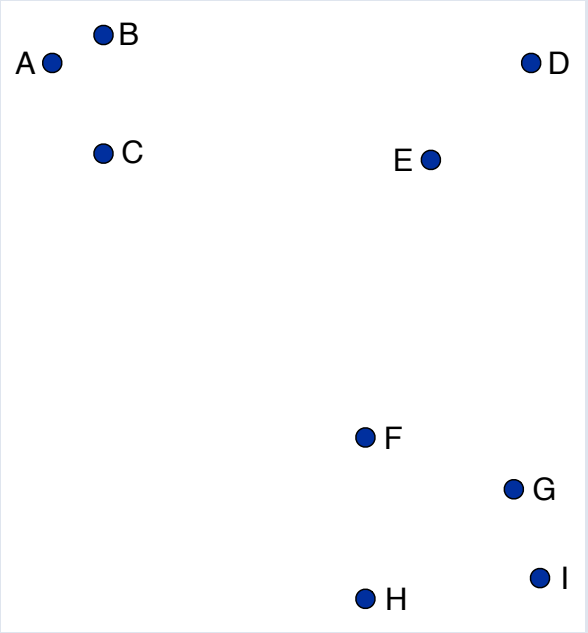



Indiana Jones, Final Fantasy, Raiders of the Lost Ark, Star Wars
Star Trek
Friends
Wallace & Gromit: The Best of Aardman Animation
The Simpsons: Season 8
The Simpsons: Season 1
The Simpsons: Season 2
Family Guy, Vol. 1: Seasons 1-2
South Park: Season 2
South Park: Bigger, Longer and Uncut
Team America: World Police
Mission: Impossible II
Mission: Impossible
The World Is Not Enough
Tomorrow Never Dies
GoldenEye
For Your Eyes Only
The Golden Gun
Goldfinger
Only Live Twice
From Russia With Love
The Spy Who Loved Me
Dr. No

Gradient interpretation

- We can interpret the t-SNE gradient as a simulation of an N-body system:

$$\frac{\partial C}{\partial \mathbf{y}_i} = 4 \sum_{j \neq i} (p_{ij} - q_{ij})(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1} (\mathbf{y}_i - \mathbf{y}_j)$$

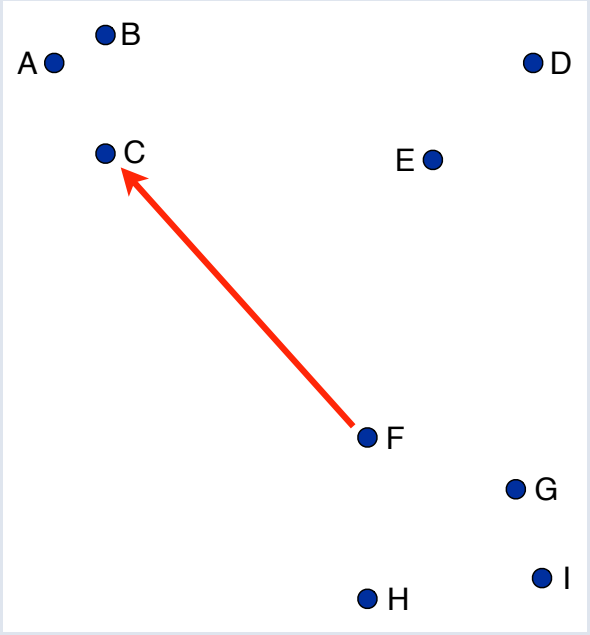


Gradient interpretation

- We can interpret the t-SNE gradient as a simulation of an N-body system:

$$\frac{\partial C}{\partial \mathbf{y}_i} = 4 \sum_{j \neq i} (p_{ij} - q_{ij})(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1} (\mathbf{y}_i - \mathbf{y}_j)$$

spring

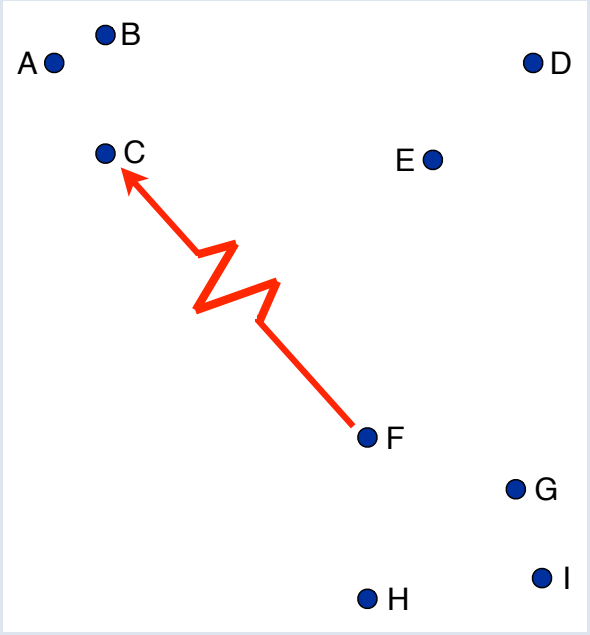


Gradient interpretation

- We can interpret the t-SNE gradient as a simulation of an N-body system:

$$\frac{\partial C}{\partial \mathbf{y}_i} = 4 \sum_{j \neq i} (p_{ij} - q_{ij})(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1} (\mathbf{y}_i - \mathbf{y}_j)$$

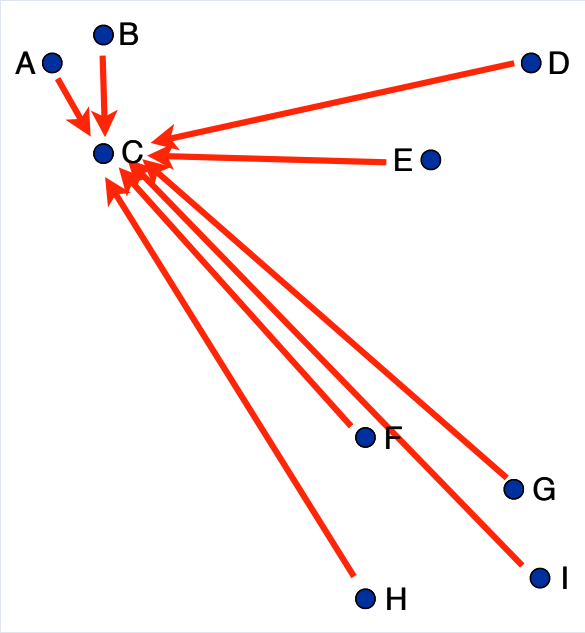
exertion / compression



Gradient interpretation

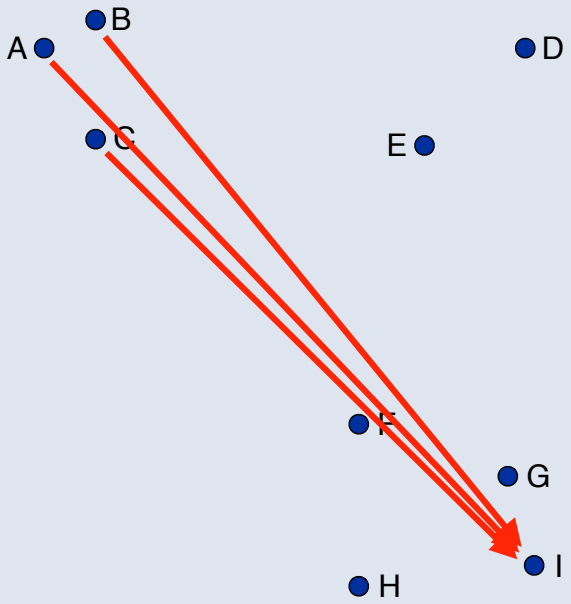
- We can interpret the t-SNE gradient as a simulation of an N-body system:

$$\frac{\partial C}{\partial \mathbf{y}_i} = 4 \sum_{j \neq i} (p_{ij} - q_{ij})(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1}(\mathbf{y}_i - \mathbf{y}_j)$$



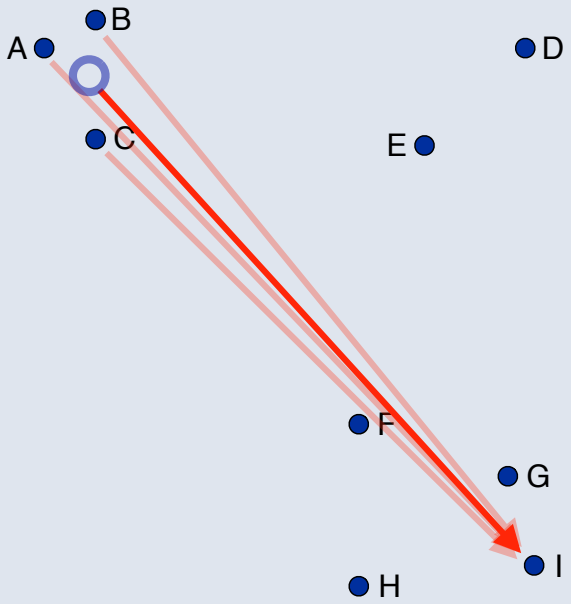
Barnes-Hut approximation

- Many of the pairwise interactions between points are very similar:



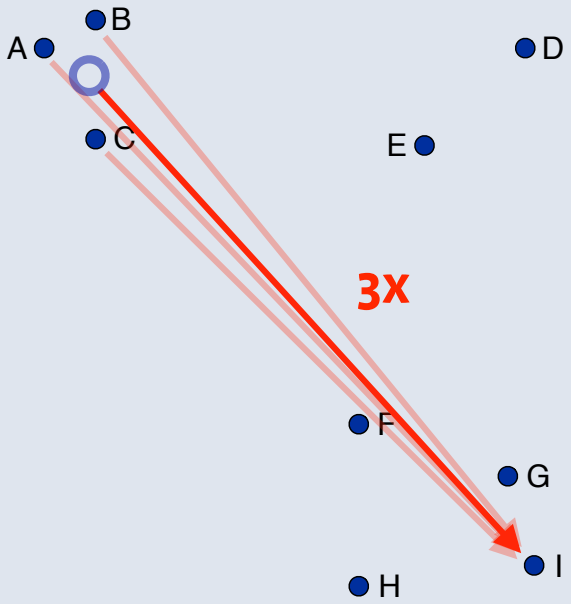
Barnes-Hut approximation

- Approximate such similar interactions by a single interaction:

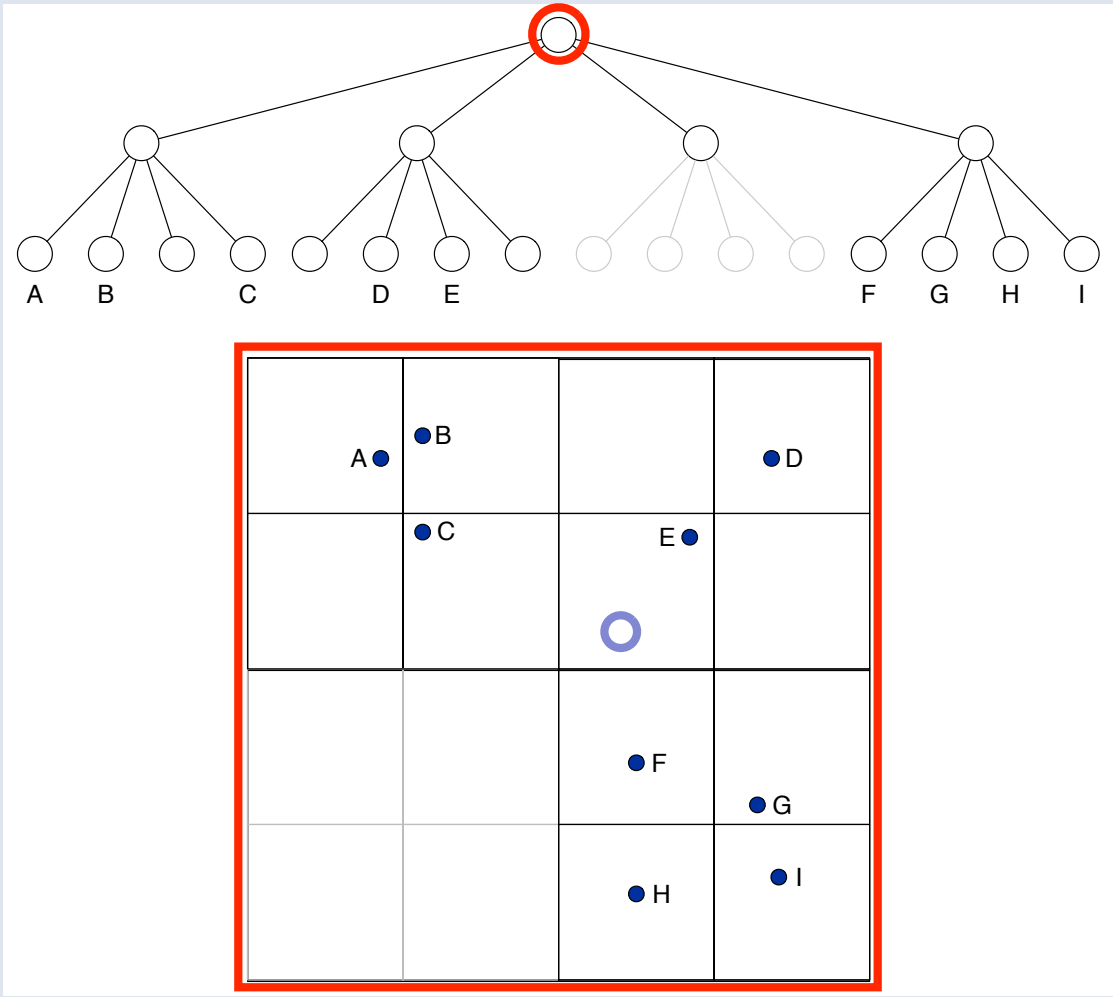


Barnes-Hut approximation

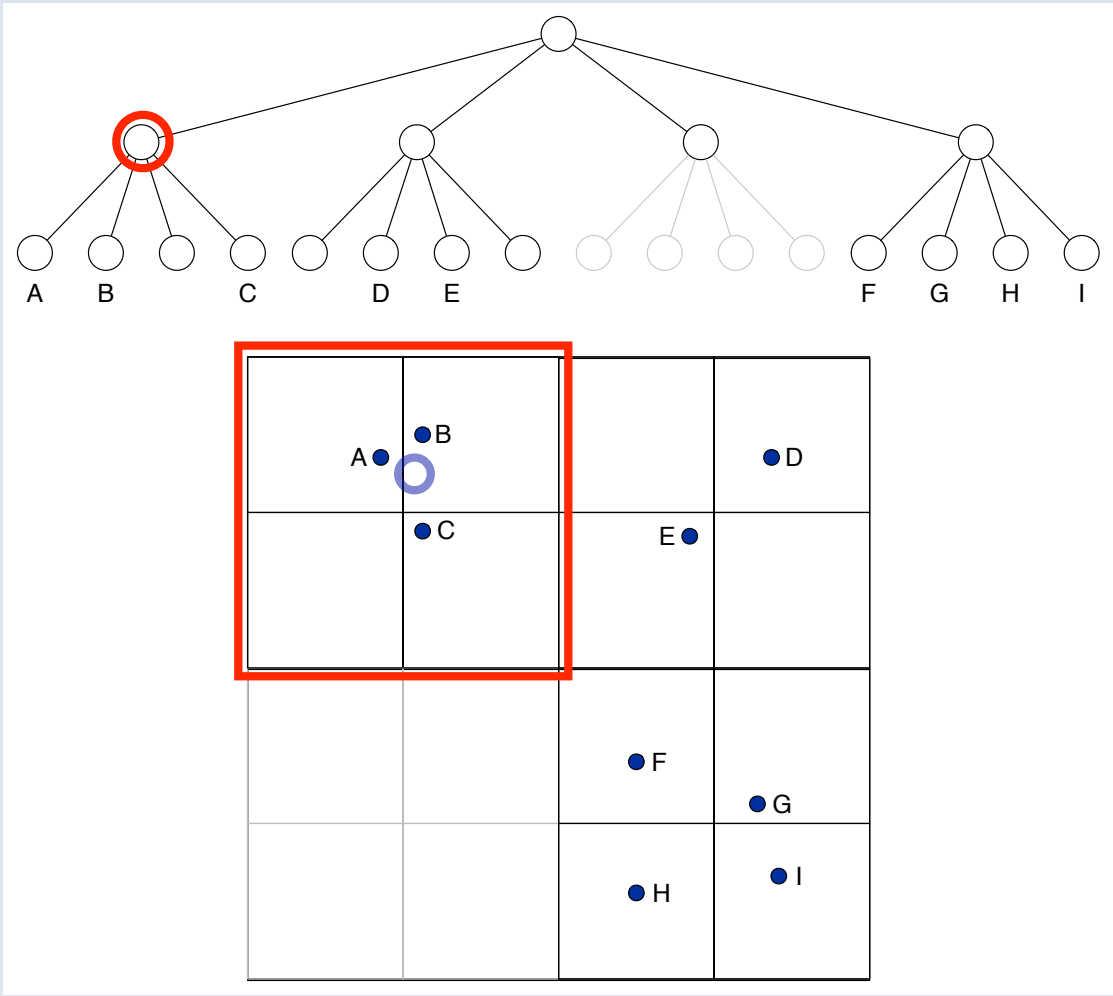
- Approximate such similar interactions by a single interaction:



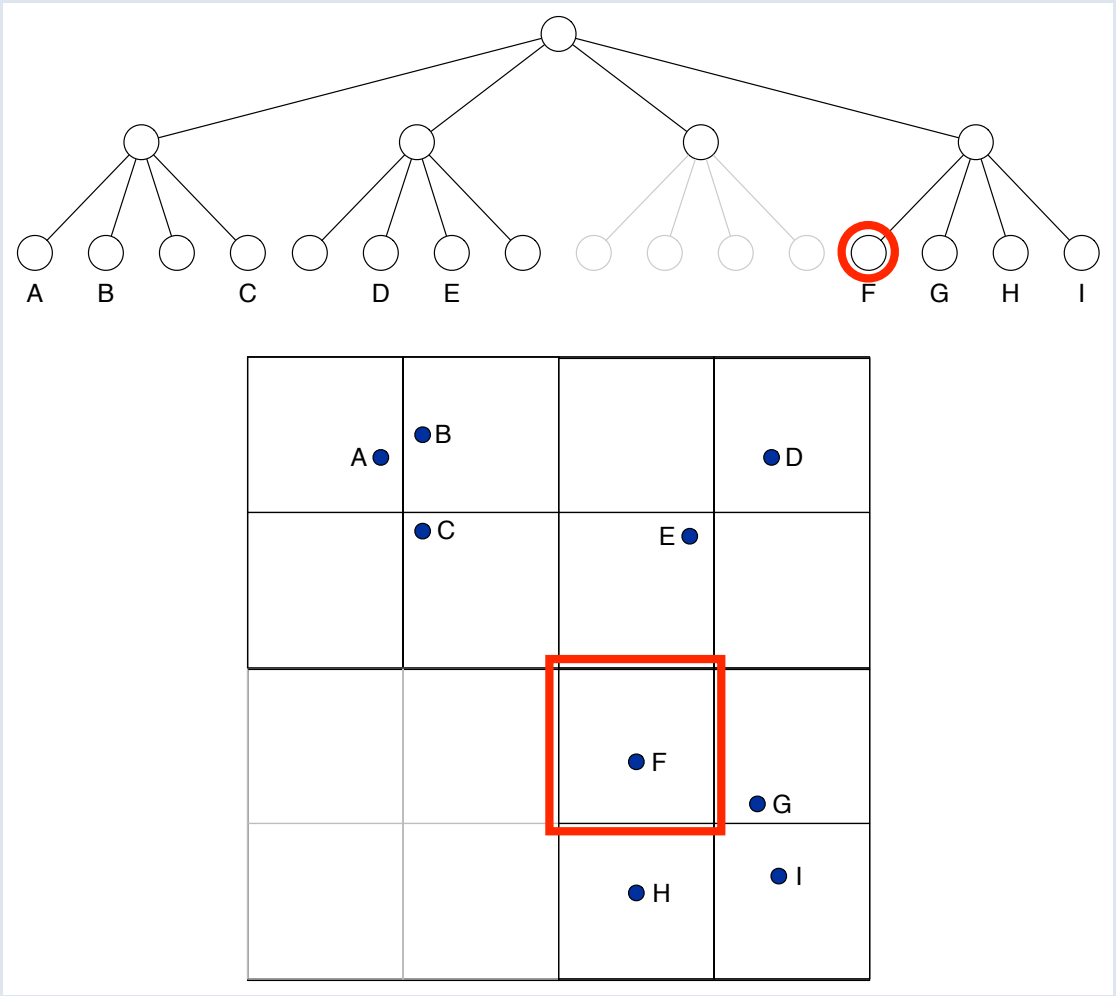
Quadtree



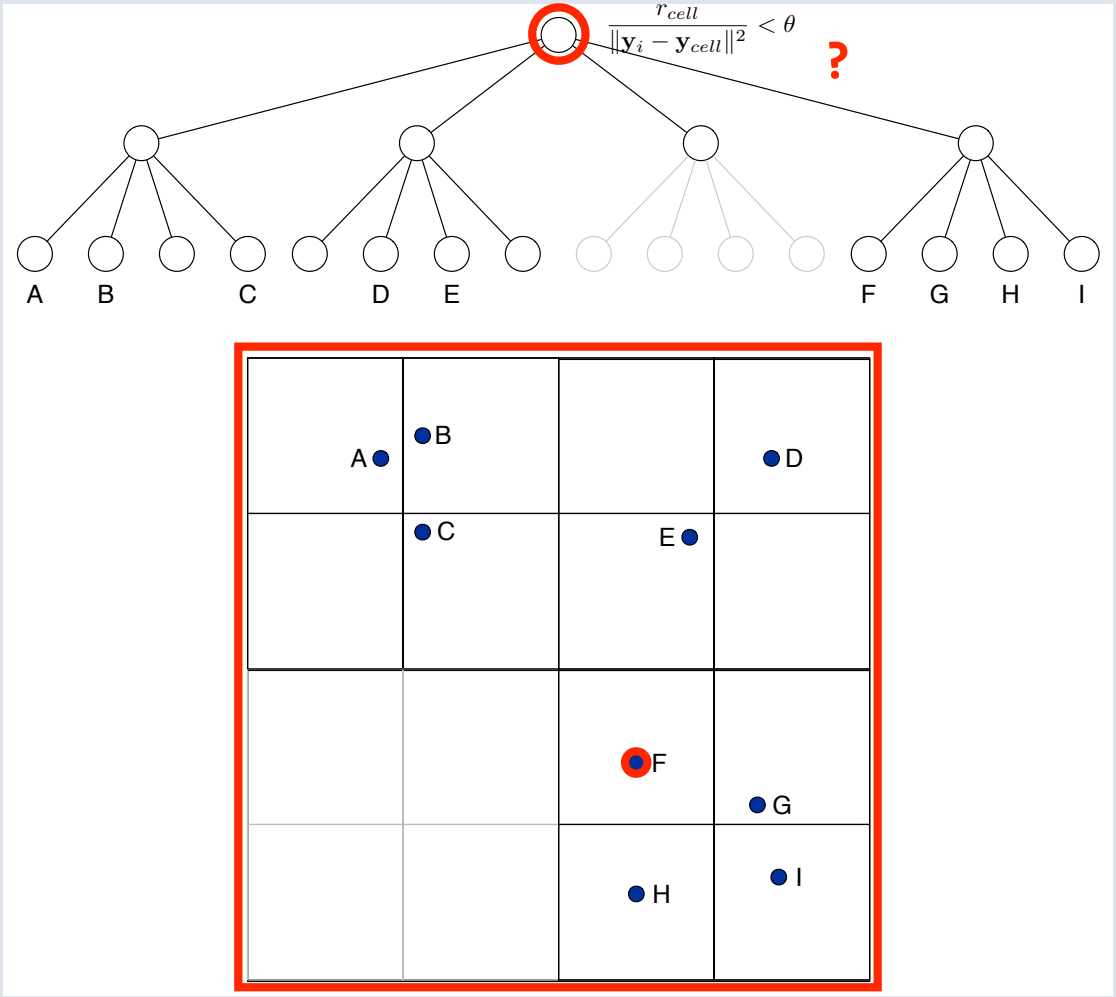
Quadtree



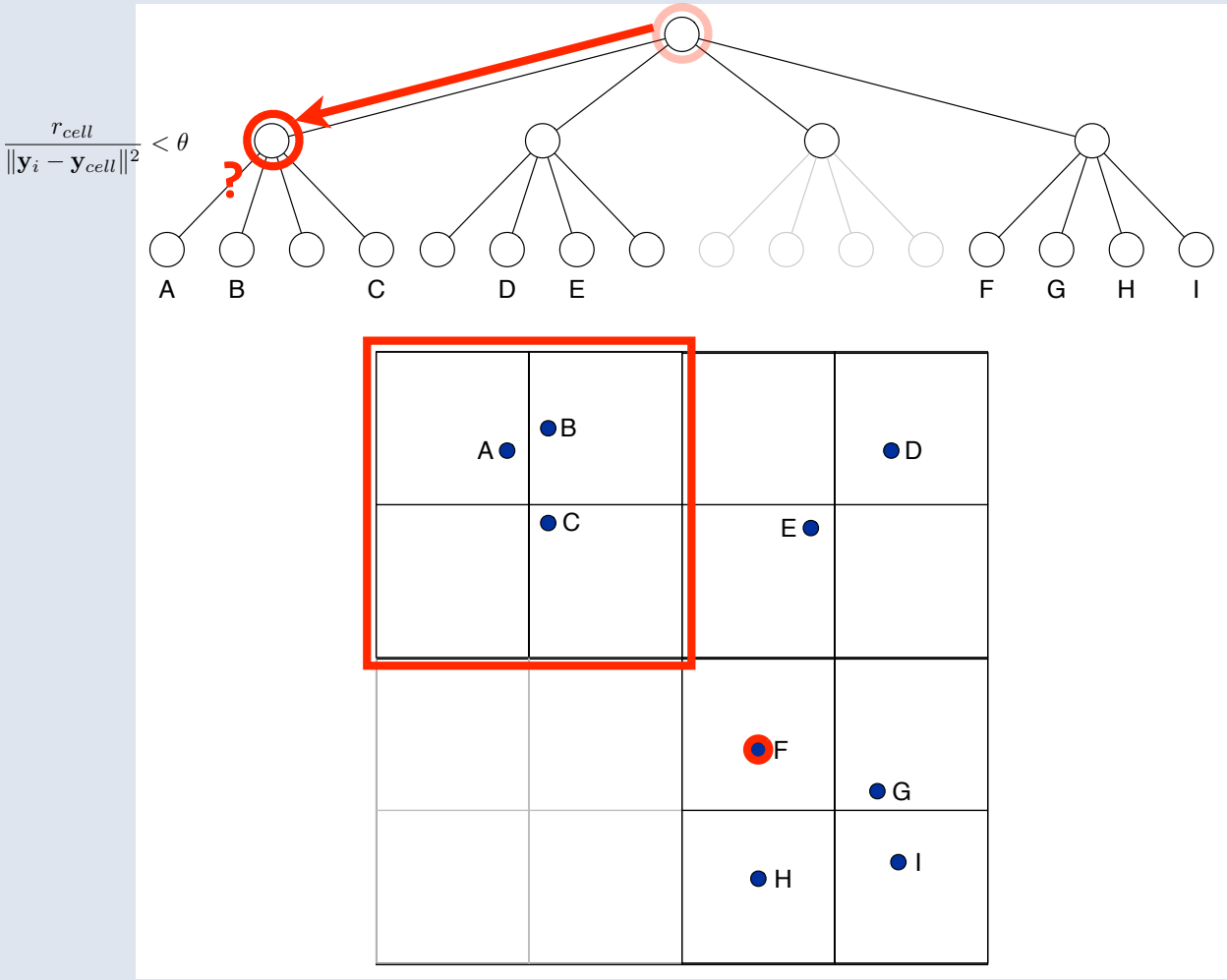
Quadtree



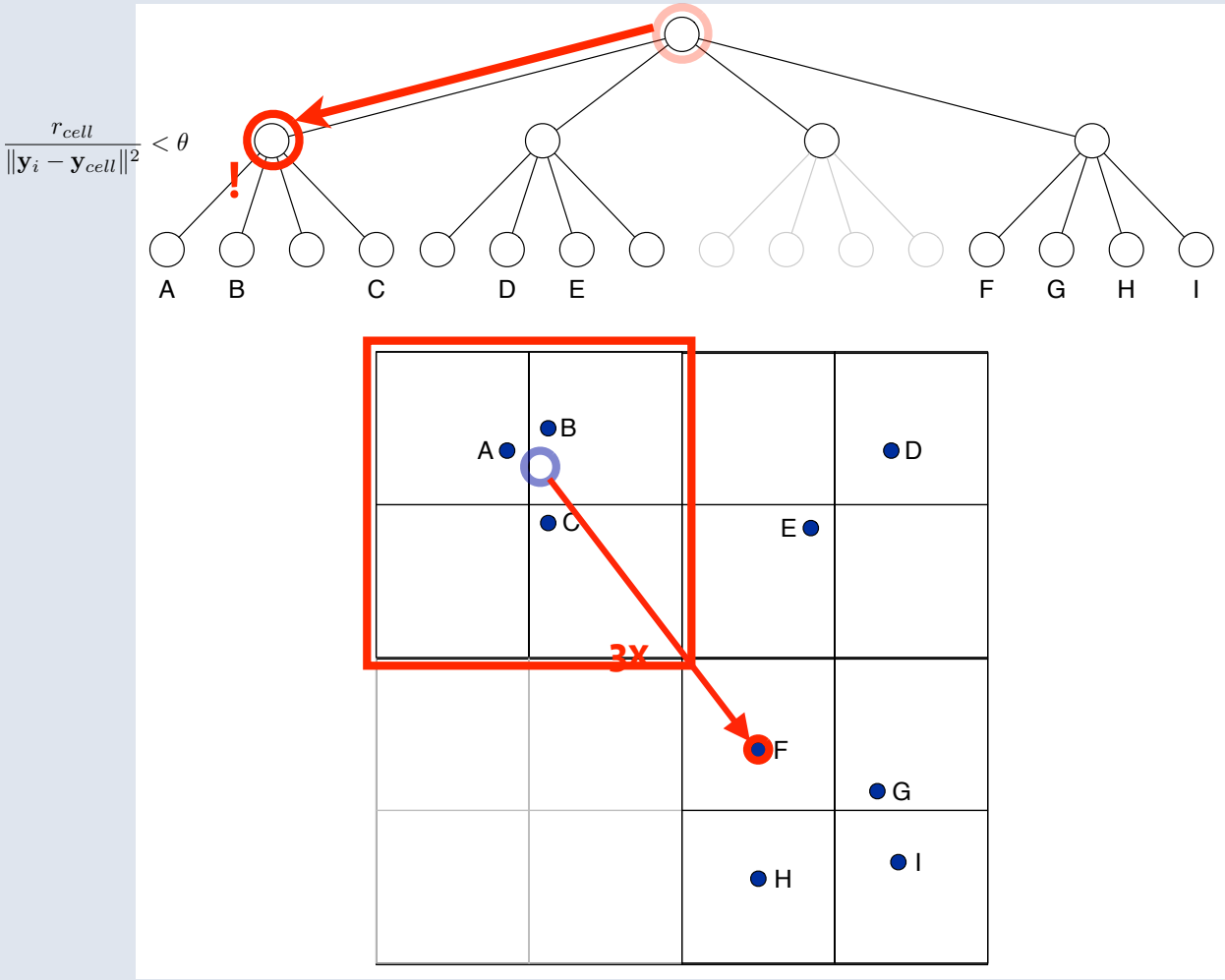
Barnes-Hut-SNE



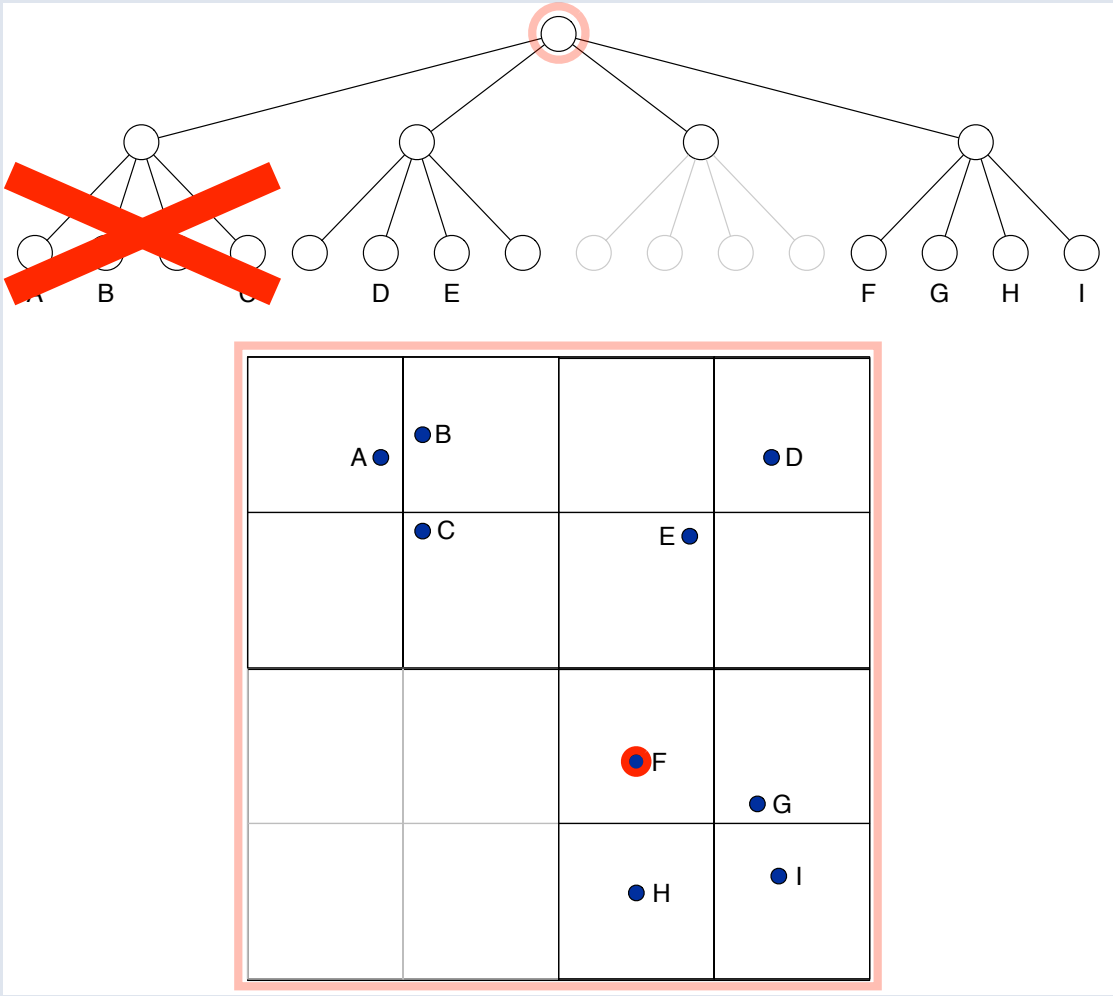
Barnes-Hut-SNE



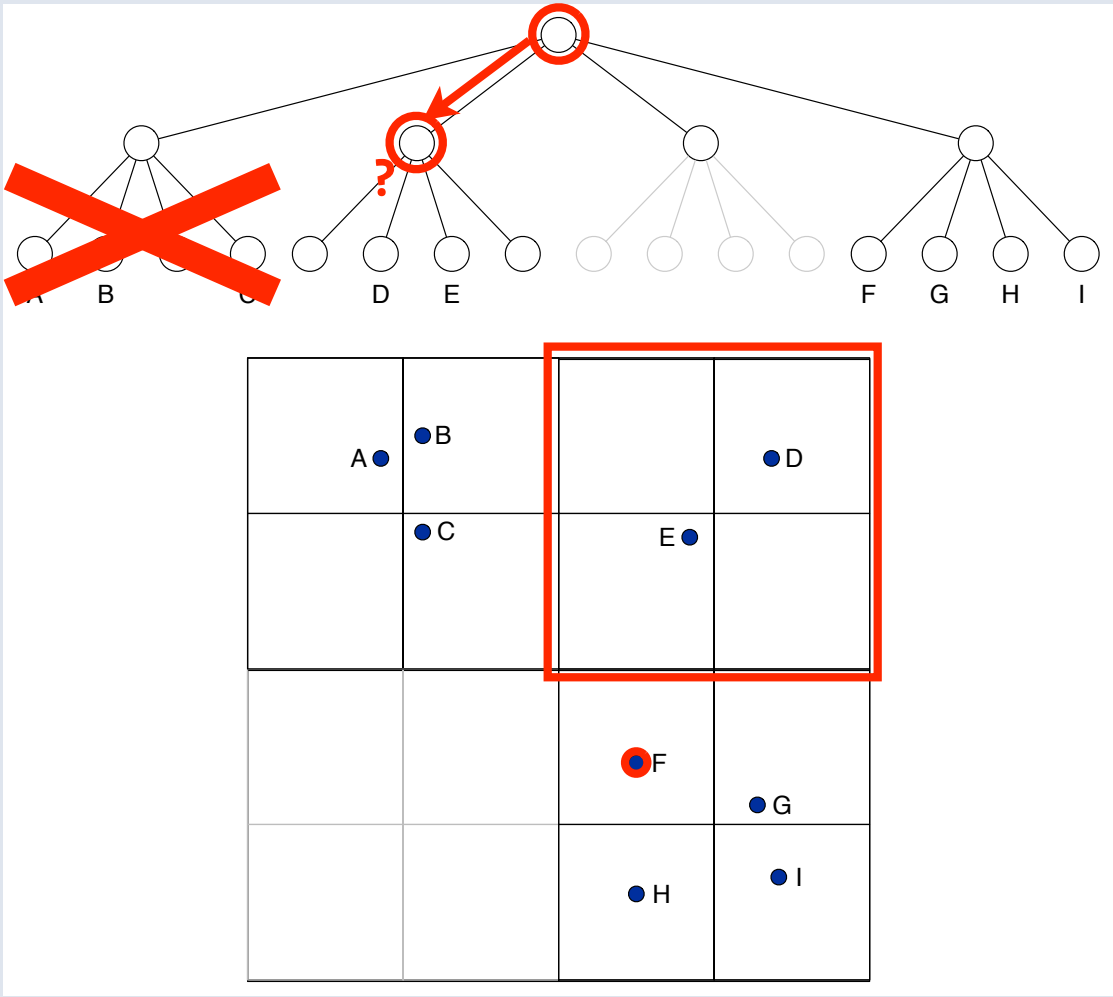
Barnes-Hut-SNE



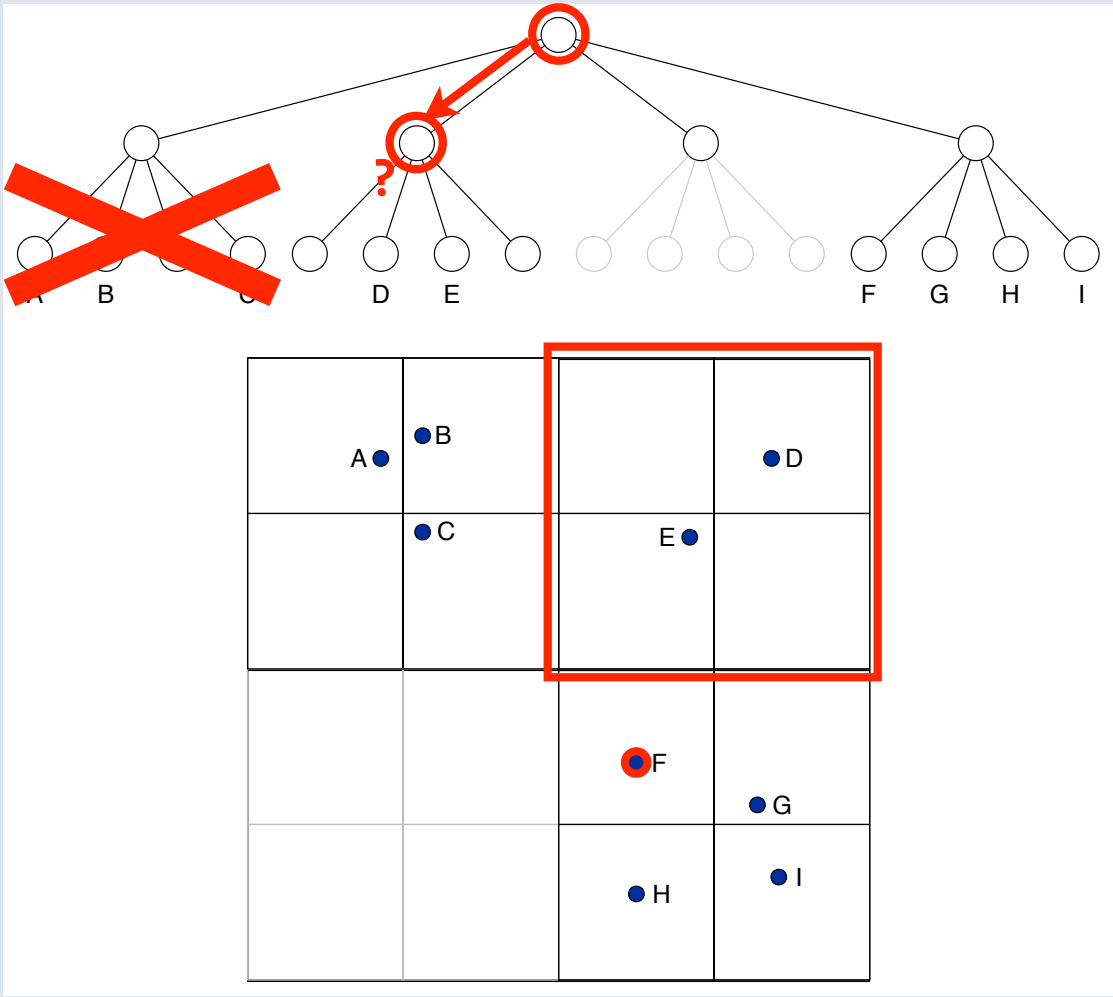
Barnes-Hut-SNE



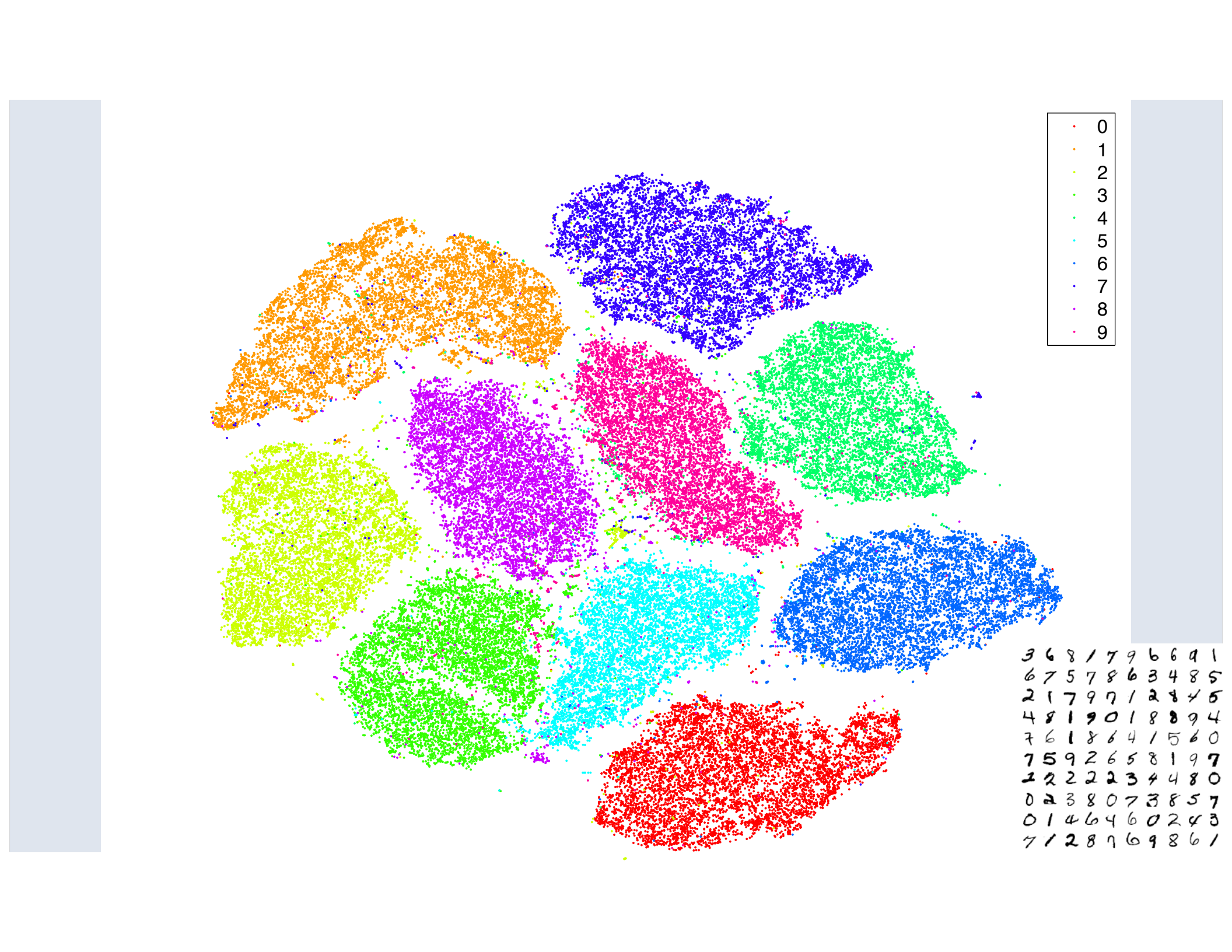
Barnes-Hut-SNE

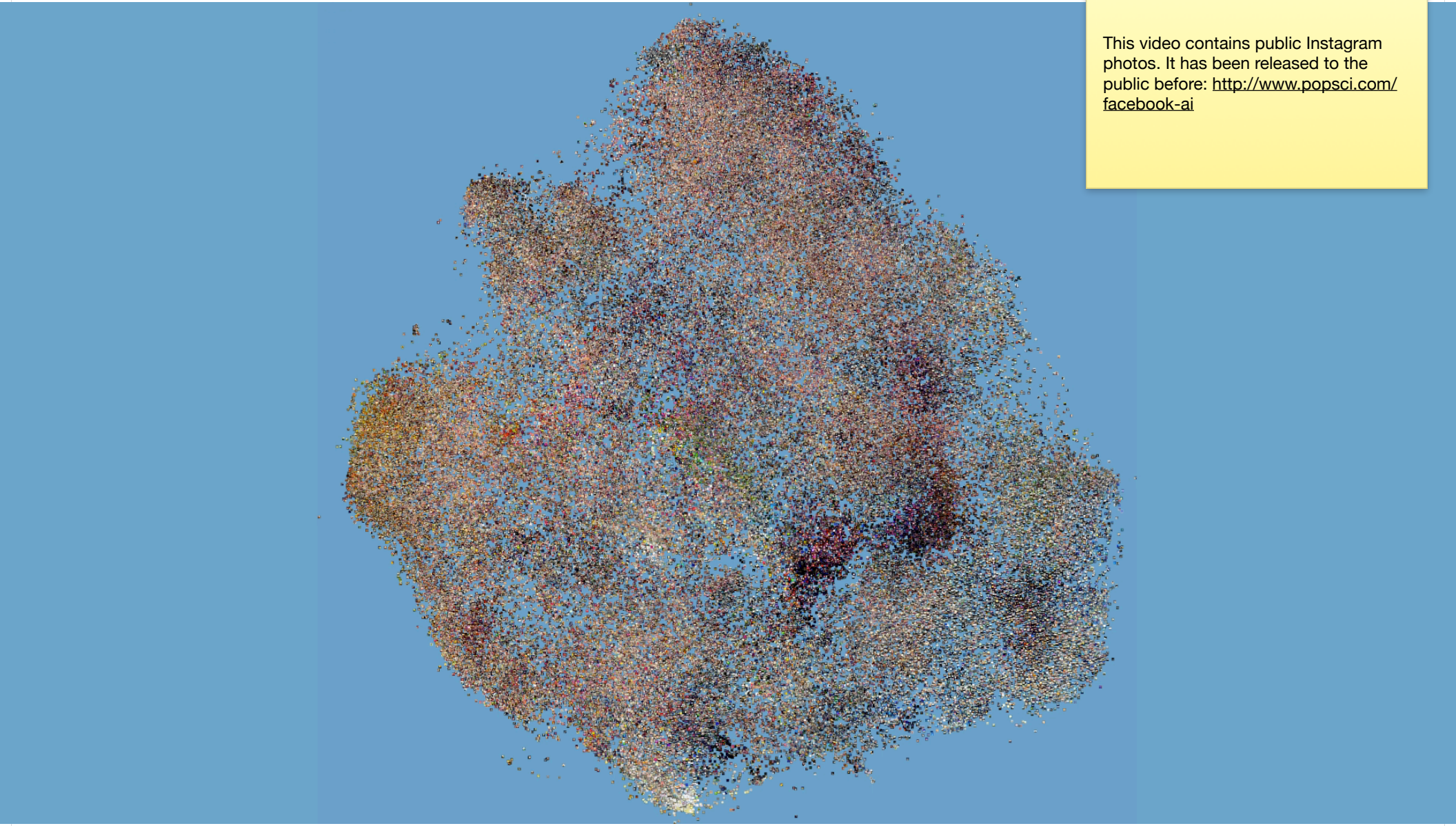


Barnes-Hut-SNE

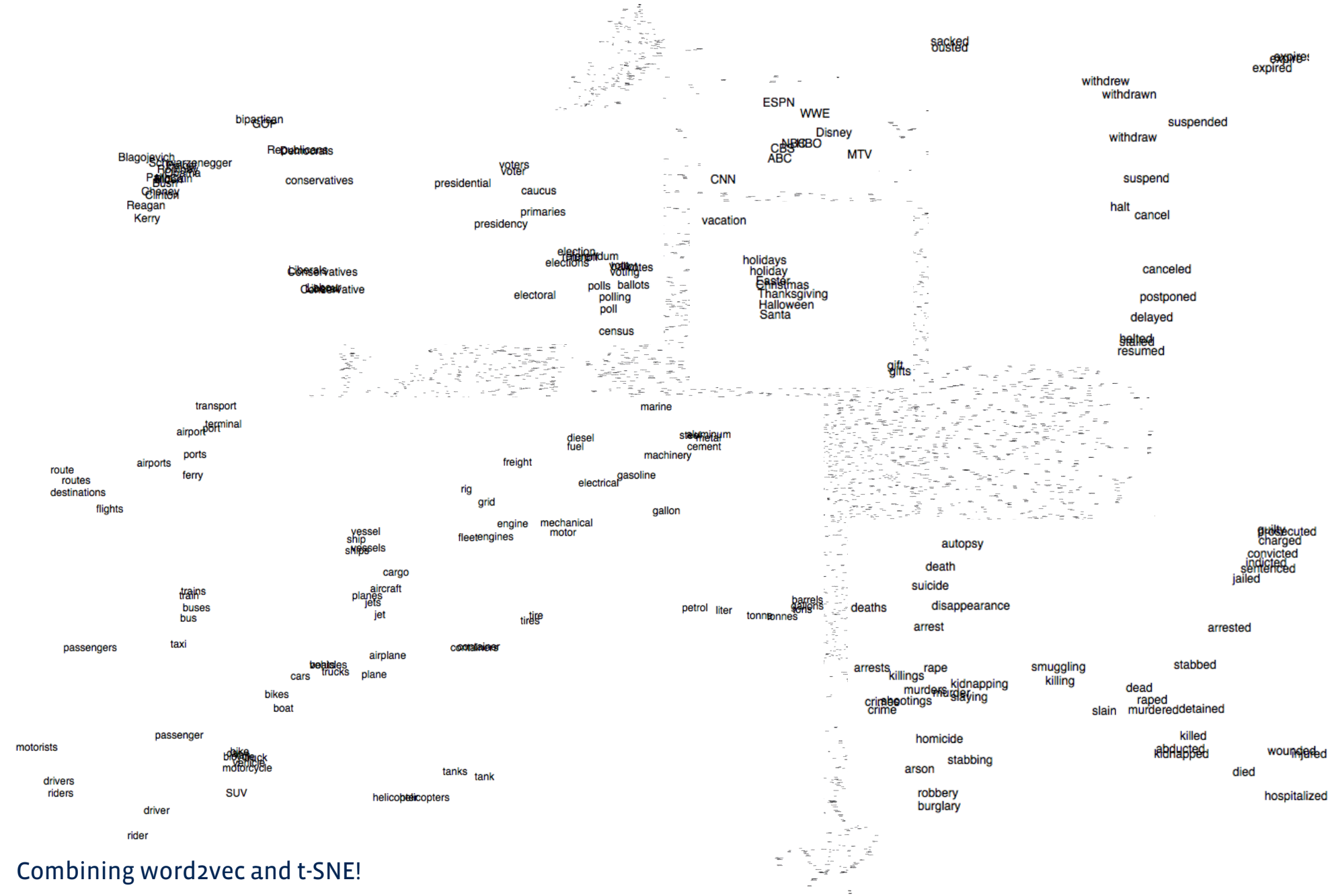


etcetera...


0
1
2
3
4
5
6
7
8
9



This video contains public Instagram photos. It has been released to the public before: http://www.popsci.com/facebook-ai



Combining word2vec and t-SNE!

Conclusions

- Embeddings provide a way for deep learners to work on discrete data
- word2vec is a popular embedding model for word representations
- t-SNE is a popular embedding model for visualization of graphs
- Many other embedding techniques exist, which differ in the exact choice for the loss function that they optimize

References

- Reading material:
  - O. Levy and Y. Goldberg. **Neural Word Embedding as Implicit Matrix Factorization**. Advances in Neural Information Processing 27:2177-2185, 2014 (first two sections).
  - L.J.P. van der Maaten and G.E. Hinton. **Visualizing High-Dimensional Data Using t-SNE**. Journal of Machine Learning Research 9(Nov):2579-2605, 2008.

- Source code:
  - Word2vec: https://code.google.com/archive/p/word2vec/
  - t-SNE: https://lvdmaaten.github.io/tsne/

facebook

Questions?